P. 1877/88

10 1988

informatyka

Dr Leslie Lamport o specyfikowaniu systemów współbieżnych

Maszyny i algorytmy równoległe

Elwro 800

WYDAWNICTWO NOT SIGMA

Cena 250 zł

informatyka

10 1988

Nr 10

Miesięcznik Rok XXIII
Październik 1988

Organ Komitetu
Naukowo-Technicznego NOT
ds. Informatyki

KOLEGIUM REDAKCYJNE:
Mgr Jarosław DEMINET,
dr inż. Wacław ISZKOWSKI,
mgr Teresa JABŁOŃSKA
(sekretarz redakcji),
Władysław KLEPACZ
(redaktor naczelny),
dr inż. Marek MACHURA,
dr inż. Wiktor RZECZKOWSKI,
mgr inż. Jan RYŻKO,
mgr Hanna WŁODARSKA,
dr inż. Janusz ZALEWSKI
(zastępca redaktora naczelnego).

PRZEWODNICZĄCY
RADY PROGRAMOWEJ:

Prof. dr hab.
Juliusz Lech KULIKOWSKI

Materiałów nie zamówionych redakcja
nie zwraca

Redakcja: 01-517 Warszawa, ul. Mickiewicza 18
m. 17, tel. 39-14-34

RSW „PRASA-KSIĄŻKA-RUCH"
PRASOWE ZAKŁADY GRAFICZNE
ul. Dworcowa 13, 85-950 BYDGOSZCZ
Zam. 3530/88. E-4
Obj. 4,0 ark. druk. Nakład 9050 egz.

ISSN 0542-9951. INDEKS 36124

Cena egzemplarza 200 zł
Prenumerata roczna 2400 zł

WYDAWNICTWO
NACZELNA ORGANIZACJA TECHNICZNA
CZASOPISM I KSIĄŻEK TECHNICZNYCH
SIGMA
00-950 Warszawa
skrytka pocztowa 1004
ul. Biała 4

W NUMERZE:

Uproszczone specyfikowanie systemów współbieżnych (1)
Leslie Lamport 1

Maszyny i algorytmy równoległe (1)
Maciej M. Sysło 5

VHDL – Ada języków opisu i projektowania sprzętu
Henryk Gizdoń, Adam Pawlak, Włodzimierz Wrona 8

System mikrokomputerowy Elwro 800 (1)
Wojciech Cellary, Jerzy Kręglewski, Ruta Maćkowiak 12

Efektywność systemu dBase III
Michał Kabsch 15

Turbo C. Modele pamięci, adresowanie, wskazania adresowe
Jan Bielecki 17

Sterowanie przepływowe (2). Implementacja modelu
Roman Podraza 20

Przegląd technik grafiki komputerowej (2)
Wojciech Mokrzycki 25

RECENZJE
Wprowadzenie do grafiki komputerowej I. O. Angella 24

TERMINOLOGIA
Terminologia systemów operacyjnych (2) 29

W NAJBLIŻSZYCH NUMERACH:

- **Henryk Gizdoń, Adam Pawlak i Włodzimierz Wrona podają pierwszą część szczegółowej charakterystyki języka opisu sprzętu VHDL, zawierającą rozwiązania wersji 7.2 tego języka.**
- **Piotr Fuglewicz i Roman Topczyński opisują program DAGGER, będący przedstawicielem nowej generacji oprogramowania dla komputerów rodziny Mera 60/660 i systemów operacyjnych RT–11, RT–60 i MTR–60.**
- **Adam Tucholski charakteryzuje zbiór procedur NETBIOS zapewniający sprzętowo niezależne sprzężenie między adapterem sieci komputerowej a systemem operacyjnym DOS.**
- **Tadeusz Czachórski i in. prezentują pakiet programowy AMOK do modelowania i oceny działania systemów komputerowych.**
- **Agnieszka Mykowiecka i Anna Ostaszewska zajmują się problematyką komunikacji z komputerem w języku naturalnym.**
- **Jan Bielecki kontynuuje serię artykułów o języku Turbo C, omawiając tym razem przerwania, sygnały i powroty.**
- **Dorota Inkielman omawia środowisko programowe Ady.**

LESLIE LAMPORT
Digital Equipment Corporation
Systems Research Center
Palo Alto, stan Kalifornia
USA

P. 1877/88

Artykuł jest dokonanym za zgodą firmy DEC tłumaczeniem raportu: L. Lamport, A Simple Approach to Specifying Concurrent Systems. Technical Report 15, DEC Systems Research Center, Palo Alto (CA), December 1986

Uproszczone specyfikowanie systemów współbieżnych (1)

W artykule przedstawiono metodę aksjomatu przejść (ang. *transition axiom method*), w której właściwości bezpieczeństwa systemu współbieżnego można wyspecyfikować za pomocą programów, a właściwości żywotności specyfikuje się przez asercje w prostej logice temporalnej. Metodę opisano za pomocą kilku prostych przykładów, a jej podstawy logiczne wyjaśniono nieformalnie badając, co oznacza zrealizowanie specyfikacji. Zagadnienia językowe i inne szczegóły, praktycznie w większości pominięto.

WPROWADZENIE

W ciągu ostatnich kilku lat rozwinąłem podejście do specyfikowania systemów współbieżnych, które obecnie nazywam metodą aksjomatu przejść. Podstawowy formalizm tej metody opisano w [1, 7], lecz szczegóły formalne przesłaniają jej istotne pojęcia. Poniżej spróbuję wyjaśnić te pojęcia bez szczegółowego omawiania ich podstaw teoretycznych.

Systemy współbieżne nie są łatwe do specyfikowania. Nawet prosty system może być bardzo subtelny i często trudno jest znaleźć odpowiednie abstrakcje, które uczynią go zrozumiałym. Specyfikowanie złożonego systemu jest ciekawym zadaniem inżynierskim. Złożone struktury można zrozumieć tylko wtedy, gdy składają się z prostych części, tak więc metoda specyfikowania złożonych systemów musi mieć proste podstawy pojęciowe. Spróbuję wykazać, że metoda aksjomatu przejść ma takie podstawy. Jednakże nie będę zajmował się zagadnieniami inżynierskimi związanymi ze specyfikowaniem rzeczywistych systemów. Zilustruję natomiast poszczególne pojęcia kilkoma łatwymi przykładami, których nie można traktować poważnie jako rzeczywistych specyfikacji.

Czy proponuję język specyfikacyjny?

Nie. Metoda aksjomatu przejść daje pojęciowe i logiczne podstawy do tworzenia specyfikacji formalnych, nie jest natomiast językiem specyfikacyjnym. Metoda określa, co powinna wyrażać specyfikacja, a język określa szczegółowo, *jak* to ma być wyrażone.

Co rozumiem przez specyfikację formalną?

Stwierdziłem, że bardzo korzystne jest rozważanie specyfikacji jako umowy (ang. *contract*) między użytkownikiem systemu a jego wykonawcą. Umowa powinna mówić użytkownikowi wszystko, co powinien on wiedzieć, aby używać systemu i mówić realizatorowi (ang. *implementer*) wszystko, co powinien on wiedzieć, aby ten system zrealizować. Z reguły, jeśli umowa zostanie uzgodniona, to użytkownik i realizator nie mają potrzeby dalszego porozumiewania się. Taki pogląd wyraża *funkcję* specyfikacji, nie stanowi natomiast paradygmatu, jak należy budować systemy.

Po to, aby specyfikacja była formalna, pytanie „Czy realizacja spełnia specyfikację?" musi być sprowadzalne do pytania „Czy asercja jest możliwa do udowodnienia w pewnym systemie matematycznym?". Aby wykazać, że spełnione są warunki umowy, realizator powinien odwoływać się do logiki, a nie do zasad prawnych. Nie znaczy to, że realizacji musi towarzyszyć dowód matematyczny. Znaczy to natomiast, że w zasadzie, choć niekoniecznie w praktyce, powinno być możliwe dostarczenie dowodu poprawności realizacji. Istnienie formalnych podstaw metody specyfikacji jest jedynym sposobem, jaki znam, aby zagwarantować, że specyfikacje są jednoznaczne.

Ponadto specyfikowane systemy są obiektami fizycznymi, a matematyka nie może dowodzić właściwości fizycznych. Można dowodzić jedynie właściwości modelu matematycznego systemu. To, czy system poprawnie realizuje model, musi pozostać zagadnieniem prawnym, a nie matematycznym.

Czym właściwie jest system?

Przez *system* rozumiem wszystko, co oddziałuje ze swoim otoczeniem w sposób dyskretny (cyfrowy) przez dobrze określoną granicę. System rezerwacji lotniczej jest takim systemem, którego granicę można poprowadzić między używającymi go operatorami, stanowiącymi część otoczenia, a terminalami stanowiącymi część systemu. Procedura w Pascalu jest systemem, którego otoczeniem jest pozostała część programu, z którą oddziałuje ona odpowiadając na wywołania i używając zmiennych globalnych. Tak więc, specyfikowany system może być składnikiem innego, większego systemu.

System słoneczny nie jest systemem w tym sensie, ponieważ nie jest systemem dyskretnym i nie ma dobrze określonego pojęcia otoczenia, z którym on oddziałuje.

Rzeczywiście systemy mają wiele istotnych właściwości, od czasu reakcji (ang. *response time*) *aż do koloru szaf. Żadna metoda formalna nie pozwala na specyfikowanie wszystkich tych właściwości. Które z nich mogą być specyfikowane za pomocą metody aksjomatu przejść?*

Metoda aksjomatu przejść specyfikuje zachowanie systemu, tzn. ciąg obserwowalnych akcji wykonywanych przez system podczas oddziaływania z otoczeniem. Mówiąc dokładniej, określa ona dwie klasy właściwości dotyczących zachowania systemu: właściwości bezpieczeństwa i żywotności. Właściwości bezpieczeństwa określają, co system może robić lub – równoważnie – czego nie może robić. Częściowa poprawność jest przykładem właściwości bezpieczeństwa zapewniającej, że program nie może generować niepoprawnych odpowiedzi. Właściwości żywotności określają, co system musi robić. Terminalność jest przykładem właściwości żywotności zapewniającej, że program musi ostatecznie wygenerować odpowiedź. Te dwie klasy właściwości zdefiniowali formalnie Alpern i Schneider [2]. W metodzie aksjomatu przejść właściwości bezpieczeństwa i żywotności są specyfikowane oddzielnie.

Niektóre istotne właściwości dotyczące zachowania nie mogą być specyfikowane metodą aksjomatu przejść. Należą do nich m. in. średni czas reakcji i prawdopodobieństwo awarii (ang. *probability of failure*). Metodą aksjomatu przejść można zbudo-

wać formalny model służący do analizy takich właściwości [1], ale nie można ich formalnie specyfikować.

Istnieją również ważne właściwości systemów nie dotyczących zchowania, jak np. wymagania pamięciowe lub kolor szaf, które niekiedy należałoby specyfikować. Leżą one jednak całkowicie poza zakresem tej metody.

Dlaczego właściwości bezpieczeństwa i żywotności specyfikuje się oddzielnie?

Ponieważ metoda aksjomatu przejść jest oparta na jednolitym formalizmie, nie ma formalnego rozdziału między specyfikacją właściwości bezpieczeństwa i żywotności. Jednakże, jak wykazuje doświadczenie, do dowodzenia obu rodzajów właściwości stosuje się różne techniki, dlatego w praktyce wygodnie jest je rozdzielić. Uważam, że zdolność do dekomponowania specyfikacji na właściwości bezpieczeństwa i żywotności jest jedną z zalet tej metody. W celu zweryfikowania właściwości żywotności należy dowieść właściwości bezpieczeństwa, lecz ten dowód można zdekomponować na prostsze lematy.

Czy tą metodą można specyfikować zachowanie w czasie rzeczywistym?

Można specyfikować najgorsze przypadki, ponieważ wymaganie, aby system reagował w określonym odcinku czasu, można wyrazić jako właściwości bezpieczeństwa, mianowicie, że zegar nie może osiągnąć określonej wartości, zanim nie uzyska się odpowiedzi systemu. Średni czas reakcji nie może być wyrażony jako właściwości bezpieczeństwa lub żywotności.

Metodą aksjomatu przejść można zapewnić, że pewne akcje muszą wystąpić (żywotność) lub nie mogą wystąpić (bezpieczeństwo). Czy można również zapewnić, że określona akcja może wystąpić?

Nie. Specyfikacja narzuca ograniczenia kontraktowe na zachowanie systemu. Asercja, że system może coś zrobić lub może nie zrobić, nie stanowi ograniczenia i dlatego nie odgrywa żadnej roli jako część specyfikacji formalnej. W metodach specyfikacji, w których stosuje się takie asercje, służą one za substytuty właściwości żywotności. Niektórymi metodami nie można wyspecyfikować właściwości, że określony sygnał wejściowy *musi* spowodować określoną reakcję, zamiast tego specyfikuje się, że jest możliwe, aby po określonym sygnale wejściowym wystąpiła określona reakcja. Każda specyfikacja, z jaką się zetknąłem, używająca takich asercji, była ulepszona przez ich zastąpienie właściwościami żywotności, które dokładniej wyrażają nieformalne wymagania nakładane na system.

Nieprecyzyjne wyrażanie się może sprawić wrażenie, że specyfikacja zawiera asercję możliwości, podczas gdy w rzeczywistości jej nie zawiera. Przykładowo, często żąda się, aby były dopuszczalne straty wiadomości w linii transmisyjnej. Jednakże w specyfikacji nie wymaga się, aby straty wiadomości były dopuszczalne, ponieważ wykluczałoby to realizację, która gwarantuje transmisję bez strat wiadomości. Specyfikacja może wymagać, aby coś się stało (właściwość żywotności) lub coś się nie stało (właściwość bezpieczeństwa) bez względu na straty wiadomości. Stwierdzenie, że wiadomości mogą być tracone, może być także komentarzem do specyfikacji nie będącym jej częścią, wyrażającym fakt, że nie wymaga się dostarczania wszystkich wiadomości.

Jeśli właściwość bezpieczeństwa zapewnia, że pewna akcja nie może wystąpić, to czy jej negacja zapewnia, że ta akcja jest możliwa?

W systemie logicznym należy odróżnić formułę logiczną **A** od asercji $\vdash$**A**, która oznacza, że **A** jest możliwe do udowodnienia w tej logice. Sama asercja $\vdash$**A** nie jest formułą tej logiki. W logice leżącej u podstaw metody aksjomatu przejść, jeżeli **A** reprezentuje właściwość bezpieczeństwa, to negacja **A**, tzn. formuła – **A**, zapewnia, że ta akcja musi wystąpić. Możliwość akcji jest wyrażona przez negację $\vdash$**A**, które jest metatwierdzeniem, a nie formułą w tej logice (por. [6]).

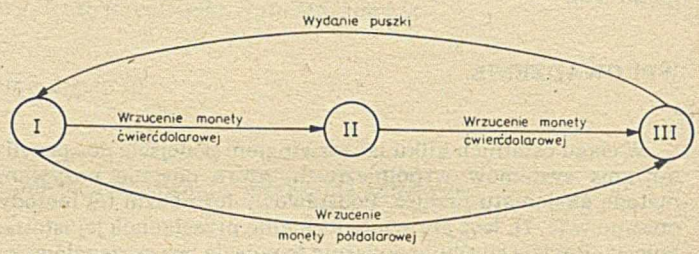
[1] Por. przykład analizy awarii zastosowany do specyfikacji [11].

WŁAŚCIWOŚCI BEZPIECZEŃSTWA

W pierwszej części przedstawię specyfikację automatu z wodą sodową.

Automat z wodą sodową

Pierwszy przykład dotyczy automatu, który wydaje puszkę z wodą sodową po wrzuceniu monety półdolarowej lub dwóch monet ćwierćdolarowych. Na rysunku przedstawiono prostą specyfikację właściwości bezpieczeństwa tego automatu uwzględniając warunek początkowy, że automat rozpoczyna pracę w stanie I. Z rysunku wynika, że będąc w stanie I automat może podjąć akcję związaną z *wrzuceniem monety ćwierćdolarowej* i przejść do stanu II lub – związaną z *wrzuceniem monety półdolarowej* i przejść do stanu III. W stanie III może wystąpić jedynie akcja *wydania puszki*, przeprowadzająca automat do stanu I. Jest to specyfikacja bezpieczeństwa, tzn. zapewniająca, że dopuszczalne jest wystąpienie tylko wymienionych akcji. Nie zapewnia ona jednak, że jakiekolwiek akcje muszą wystąpić.



Specyfikacja automatu z wodą sodową

Co się stanie, jeśli użytkownik wrzuci najpierw monetę ćwierćdolarową, a potem półdolarową?

Specyfikacja nie dopuszcza takiego zachowania. Należy bowiem pamiętać, że dobrane przykłady nie są realistycznymi specyfikacjami. Istnieją dwa sposoby rozważenia tego aspektu specyfikacji.

- Specyfikacja ogranicza zachowanie użytkownika, zabraniając mu wrzucenia monety półdolarowej po wrzuceniu ćwierćdolarowej.
- Specyfikacja nie określa, co uczyni automat, jeśli użytkownik wrzuci najpierw monetę ćwierćdolarową, a później półdolarową. Realizatorowi pozostawia się swobodę co do konstrukcji automatu reagującego na ten rodzaj „niewłaściwego" zachowania użytkownika.

Wybór jednego z tych podejść nie ma wpływu na to, jak się pisze specyfikacje i jak się ich dowodzi.

Rysunek miał specyfikować zachowanie automatu z wodą sodową. Dlaczego zatem specyfikuje on także zachowanie użytkownika?

Nie można zrealizować systemu, który funkcjonuje właściwie wobec dowolnego zachowania otoczenia. Bardziej realistyczna specyfikacja będzie pozwalała użytkownikowi na wrzucenie monet w dowolnej kolejności, na przykład, zakładając w takim przypadku ich zwrot przez automat. Nie będzie jednak pozwalała użytkownikowi na zaatakowanie automatu młotkiem (później stanie się jasne, w jaki sposób). Specyfikacja procedury zawiera zwykle warunek początkowy (ang. *precondition*), ograniczający zachowanie otoczenia przez zabronienie wywołań, których argumenty nie spełniają tego warunku. Specyfikacja obwodu elektronicznego zawiera ograniczenia czasowe, określające kiedy otoczenie może zmienić wartości sygnałów wejściowych [10].

Przedstawiony rysunek jest prostym diagramem przejść stanów (ang. state transition diagram). Takie diagramy są użyteczne dla bardzo prostych przykładów, ale czy nie skomplikują się zanadto przy specyfikowaniu rzeczywistych systemów?

Tak, tych diagramów nie buduje się zbyt łatwo dla większych problemów. Diagramy przejść stanów stanowią tylko jeden szczególny język, którego można używać w metodzie aksjomatu przejść. Wady tych diagramów są ograniczeniami samego języka, a nie – metody aksjomatu przejść. Do pisania specyfikacji większych systemów tą metodą potrzebne są inne języki. Do kwestii języków wrócę jeszcze w dalszej części artykułu.

Jakie jest zatem fundamentalne, niezależne od języka pojęcie wyrażone przez diagram przejść stanów przedstawiony na rysunku?

Są to dopuszczalne zmiany stanów. W metodzie aksjomatu przejść specyfikuje się właściwości bezpieczeństwa opisując zbiór stanów i wszystkie dopuszczalne przejścia między stanami. Istnieje wiele różnych języków, za pomocą których można opisywać stany i przejścia między nimi.

Pojęcie zmiany stanów zilustrowane przez diagram na rysunku jest używane od wielu lat. Czy metoda aksjomatu przejść zawiera cokolwiek ponadto?

Nowością w metodzie aksjomatu przejść nie jest sam diagram, lecz jego interpretacja jako specyfikacji formalnej. Ta nowa interpretacja jest potrzebna, ponieważ konwencjonalne metody przejść stanów nie zajmują się właściwie pytaniem, co znaczy dla realizacji spełnienie takiej specyfikacji. Jedną z zalet metody aksjomatu przejść jest to, że specyfikacje właściwości bezpieczeństwa można pisać za pomocą przyjaznej, zrozumiałej notacji, jak np. diagramy przejść stanów. Choć specyfikacje nie wyglądają na nowe, nowe jest znaczenie jakie im się przypisuje [2].

Co nowego jest w interpretacji rysunku za pomocą metody aksjomatu przejść?

Konwencjonalna interpretacja rysunku stwierdza, że specyfikuje on automat trójstanowy. W metodzie aksjomatu przejść uważa się, że istnieje pewna funkcja stanu, powiedzmy **f**, która przybiera wartości I, II i III, a diagram specyfikuje jak zmienia się funkcja. Mówiąc dokładniej, zakłada się, że automat z wodą sodową ma pewien nie specyfikowany zbiór stanów, nazwany **S**, a zachowanie automatu jest opisane przez ciąg stanów $s_0, s_1, s_2, \ldots$ Funkcja Stanu **f** jest odwzorowaniem zbioru **S** na zbiór wartości {I, II, III}. Diagram z rysunku specyfikuje, że dla każdego przejścia stanów

$$s_1 \rightarrow s_i + 1$$

(w tej kolejności) zachodzi albo równość

$$f(s_i) = f(s_i + 1),$$

co może być prawdziwe nawet wtedy, gdy

$$s_1 \neq s_i + 1$$

albo jedna z następujących zmian wartości z $f(s_i)$ na $f(s_i + 1)$:

- $f(s_i) = I$, $f(s_i + 1) = II$, a zmiana jest spowodowana przez *wrzucenie monety ćwierćdolarowej*
- $f(s_i) = I$, $f(s_i + 1) = III$, a zmiana jest spowodowana przez *wrzucenie monety półdolarowej*
- $f(s_i) = II$, $f(s_i + 1) = III$, a zmiana jest spowodowana *wrzuceniem monety ćwierćdolarowej*
- $f(s_i) = III$, $f(s_i + 1) = I$, a zmiana jest spowodowana *wydaniem puszki.*

[2] Mówiąc dokładniej, znaczenie to było nowe, gdy je zaproponowano [7, 8]. Później taka interpretacja pojawiła się też w innych pracach, np. [5].

Co zyskuje się przez tę nową interpretację?

Rzeczywisty automat z wodą sodową może mieć setki stanów wewnętrznych. Jeśli specyfikacja zapewnia, że istnieją trzy stany, to należy zadać pytanie, co to znaczy dla automatu o setkach stanów, że spełnia on specyfikację zapewniającą istnienie tylko trzech stanów. Interpretacja w metodzie aksjomatu przejść omija ten problem, ponieważ specyfikacja zapewnia, że funkcja stanu ma tylko trzy wartości, nie mówiąc nic o liczbie różnych stanów. Zalety tej interpretacji i zagadnienia wynikające z konwencjonalnej interpretacji powyższego rysunku jako automatu trójstanowego zostaną omówione dalej.

Specyfikacja formalna powinna dostarczać wszelkich informacji niezbędnych do określenia, czy realizacja jest poprawna. Jednakże, z rysunku w żaden sposób nie wynika, czy realizacja ma składać się z:
1) całego urządzenia włącznie z pojemnikiem na monety i stojakiem na puszki,
2) układu elektronicznego wenątrz maszyny,
3) programu dla mikroprocesora sterującego.
Dobór etykiet na łukach diagramu może dostarczać pewnych wskazówek, lecz z pewnością nie może mieć żadnego znaczenia formalnego. Czy zarówno pełne urządzenie z wodą sodową, układ elektroniczny, jak i program komputerowy mogą być poprawnymi realizacjami tej samej specyfikacji formalnej?

Specyfikacja musi być niepełna, jeśli nie rozróżnia między urządzeniem mechanicznym, układem elektronicznym a programem. Na rysunku brakuje specyfikacji *sprzężenia* (ang. *interface*), tj. mechanizmu, za pomocą którego system komunikuje się z otoczeniem. Specyfikacja musi określać, czy komunikacja odbywa się przez wrzucanie monet i wydawanie puszek, czy przez zmiany poziomu napięć na przewodach, czy przez wywoływanie i powroty z procedur programowych. Specyfikacja sprzężenia określa, że *wrzucenie monety* nie może być wykonane (zastąpione) młotkiem. Różnica między wrzuceniem monety a użyciem młotka, podobnie jak między podaniem sygnału o napięciu 5V a sygnału o napięciu 5kV, może być opisana tylko z użyciem szczegółów realizacyjnych. Sprzężenie musi być zatem specyfikowane na poziomie realizacji.

Jak specyfikuje się sprzężenie?

System i jego otoczenie komunikują się przez zmiany wartości funkcji stanu. Przykładowo, jeśli specyfikuje się układ elektroniczny, to komunikacja między tym układem a jego otoczeniem odbywa się przez zmiany wartości funkcji stanu reprezentujące różne wartości napięć na poszczególnych liniach[3]. Każdej linii **w** może odpowiadać funkcja stanu f_w, która reprezentuje napięcia na tej linii. Specyfikacja może zezwalać otoczeniu na komunikowanie się z układem przez zmianę wartości funkcji f_w na $4{,}5 \pm 1{,}2$ V, tzn. ustaloną wartość napięcia na linii między 3,3 V a 5,7 V. Innej linii **w'** mogą odpowiadać wartości funkcji f_w, z przedziału $0 \pm 1{,}2$ V. Mimo ciągłego zakresu wartości napięcia, system można nadal uważać za dyskretny, ponieważ zmiany napięcia uważa się za chwilowe.

Aby można było specyfikować sprzężenia, należy określić w specyfikacji, jak zmieniają się takie funkcje stanu sprzężenia. Można to zrobić tą samą metodą, której użyto do specyfikowania zmian wewnętrznych funkcji stanu, takich jak funkcja **f** w specyfikacji automatu z wodą sodową. Tak więc, metoda aksjomatu przejść nie wymaga żadnego dodatkowego aparatu do specyfikowania sprzężenia.

W praktyce zwykle nie specyfikuje się sprzężenia w ten sposób. Sprzężenie jest specyfikowane w języku realizacji, co powoduje, że łatwiejsze jest sprawdzenie poprawności realizacji. Przykładowo, linie łączące układ elektroniczny z otoczeniem mogą być wyspecyfikowane bezpośrednio w języku opisu sprzętu użytym do realizacji układu. Zamiast specyfikować, jak zmieniają się rzeczywiste wartości napięcia na linii **w**, można

[3] W rzeczywistości specyfikuje się nie układ elektroniczny, lecz model matematyczny tego układu. Funkcje stanu są matematycznymi obiektami tego modelu, które reprezentują napięcia na liniach rzeczywistego układu.

opisać te zmiany używając elementarnych operacji języka opisu sprzętu, np.:

w: = true

bez podawania rzeczywistych wartości.

Jak specyfikuje się sprzężenia programowe?

Istota specyfikacji sprzężenia zależy od stosowanego języka programowania. Dla procedury w Pascalu sprzężenie specyfikuje się podając nazwę procedury, typy jej argumentów oraz nazwy i typy wszystkich zmiennych globalnych, do których procedura ma dostęp. Dla pakietu Moduli 2 sprzężenie specyfikuje się w module definicyjnym [12].

Znaczy to, że nie można specyfikować procedury niezależnie od języka, w którym jest zrealizowana. Czy nie ma zatem możliwości napisania specyfikacji, powiedzmy funkcji obliczającej pierwiastki kwadratowe, która jest niezależna od języka realizacji?

Nie specyfikuje się języka, w którym procedura ma być zrealizowana, lecz specyfikuje się realizację sprzężenia. System, którego sprzężenie jest wyspecyfikowane jako procedura pascalowa, może być zrealizowany w języku asemblera. Powinien jedynie odpowiadać tym samym konwencjom wywoływania, co procedura w Pascalu.

Choć specyfikacje funkcji obliczającej pierwiastki kwadratowe będą podobne dla różnych języków programowania, nie będą jednak identyczne. Przykładowo, sposób obsługi błędów będzie zależał od tego, czy dany język ma odpowiednie mechanizmy. Rozdzielenie specyfikacji funkcji obliczającej pierwiastki kwadratowe na część wspólną i część należącą do sprzężenia, jest zagadnieniem dotyczącym języka specyfikacyjnego, rozważanym w [3].

Wpływ sprzężenia na resztę specyfikacji jest szczególnie istotny w systemach współbieżnych. W artykule [9] wykazano, że specyfikacja nawet tak prostej właściwości jak priorytet FIFO nie może być niezależna od szczegółów realizacji sprzężenia.

Wynika stąd, że nawet dla specyfikacji na najwyższym poziomie sprzężenie należy specyfikować na poziomie realizacji. Czy nie można ukryć niskopoziomowych szczegółów realizacyjnych w specyfikacji na wysokim poziomie?

Sprzężenie specyfikuje się opisując, jak mogą zmieniać się funkcje stanu sprzężenia. Poniżej pokażę, jak można specyfikować hierarchicznie zmiany wewnętrznych funkcji stanu. To samo podejście można zastosować do funkcji stanu sprzężenia. Jednakże specyfikacja wysokiego poziomu nie jest pełna dopóty, dopóki nie jest zakończone specyfikowanie sprzężenia aż do poziomu realizacji. Pełna specyfikacja powinna wyeliminować potrzebę jakiejkolwiek komunikacji między użytkownikiem systemu a jego realizatorem. Przykładowo, specyfikacja układu elektronicznego sterującego automatem z wodą sodową powinna zawierać wszystkie informacje niezbędne osobie projektującej pozostałą część urządzenia, tzn. powinna określać rzeczywiste wartości napięć na poszczególnych liniach.

O ile hierarchiczna dekompozycja sprzężenia jest użyteczna w praktyce, to logicznie jest jedynie metodą organizowania specyfikacji wysokiego poziomu. Dlatego nie będę rozważał takiej dekompozycji sprzężenia.

Czy sprzężenie można specyfikować jedynie przy użyciu funkcji stanu?

Oprócz funkcji stanu sprzężenia, trzeba wprowadzić także pojęcie odpowiedzialności za zmiany wartości tych funkcji stanu. Specyfikacja sprzężenia automatu z wodą sodową musi określać, że otoczenie (tzn. użytkownik) wykonuje *wrzucenie monety ćwierćdolarowej* lub *wrzucenie monety półdolarowej*, a system (tzn. automat) wykonuje *wydanie puszki*. Specyfikacja sprzężenia procedury musi stwierdzać, że otoczenie (tzn. pozostała część programu) wykonuje wywołanie procedury, a system (tzn. procedura) wykonuje powrót.

Zazwyczaj akcje są wykonywane albo przez otoczenie, albo przez system. Jednakże niekiedy użyteczne jest zapewnienie, że określona część systemu wykonuje akcję. Przykładowo, aby wyspecyfikować proces współdziałający ze swoim otoczeniem zarówno przez zmienne współdzielone, jak i przez operację typu CSP [4], użyteczne może być rozdzielenie akcji wykonywanych przez kanał komunikacyjny (operacje ! i ?) od wykonywanych przez inne procesy (operujące zmiennymi współdzielonymi)[4].

Dlaczego konieczne jest określenie, kto wykonuje akcje sprzężenia?

Rozważmy pakiet Moduli 2, który realizuje kolejkę dostarczając procedur **put** i **get**. Jeżeli nie poda się w specyfikacji, że tylko otoczenie może wywołać te procedury, to specyfikacja będzie spełniona przez realizację, która sama wywołuje procedurę **put**, powodując dodawanie przypadkowych elementów do kolejki.

Jaka jest ogólna postać specyfikacji właściwości bezpieczeństwa w metodzie aksjomatu przejść?

Na specyfikację właściwości bezpieczeństwa składają się następujące elementy:

- zbiór funkcji stanu rozdzielonych na wewnętrzne funkcje stanu i funkcje stanu sprzężenia,
- specyfikacja wartości początkowej każdej funkcji stanu,
- zbiór akcji rozdzielonych na akcje wewnętrzne i akcje sprzężenia,
- specyfikacja, kto wykonuje akcję sprzężenia (dla każdej takiej akcji) – akcje wewnętrzne są wykonywane przez system,
- zbiór reguł zwanych aksjomatami przejść, które określają, jak każda akcja zmienia funkcje stanu (funkcja stanu sprzężenia może być zmieniona tylko przez akcję sprzężenia).

W przykładzie automatu z wodą sodową jest jedna wewnętrzna funkcja stanu **f**, o wartości początkowej **I**. Funkcje stanu sprzężenia nie są wyspecyfikowane. Ponadto są trzy akcje będące akcjami sprzężenia: *wrzucenie monety ćwierćdolarowej* i *wrzucenie monety półdolarowej* wykonywane przez otoczenie *oraz wydanie puszki* wykonywane przez system. Skutek *wrzucenia monety ćwierćdolarowej* jest opisany przez aksjomat przejścia zapewniający, że ta akcja może wystąpić tylko wtedy, gdy **f** równa się **I**, co powoduje zmianę wartości funkcji na **II**, lub wtedy, gdy **f** równa się **II**, co powoduje zmianę jej wartości na **III**. Reguły odpowiadające *wrzuceniu monety półdolarowej* i *wydaniu puszki* są podobne, lecz nieco prostsze. Pełna specyfikacja automatu z wodą sodową musi również uwzględniać fakt, jak te trzy akcje zmieniają funkcje stanu sprzężenia.

Jakie jest dokładne znaczenie takiej specyfikacji?

Formalnym znaczeniem specyfikacji w metodzie aksjomatu przejść jest formuła logiki temporalnej. Aby podać ścisłą definicję tego znaczenia, należy zdefiniować semantykę formalną logiki temporalnej i podać algorytm tłumaczenia specyfikacji na formułę tej logiki. Uczyniono to w artykule [7]. Poniżej, zamiast takiego formalnego podejścia, spróbuję przedstawić intuicyjne rozumienie znaczenia specyfikacji metodą aksjomatu przejść, przez szczegółowe rozważenie, co znaczy zrealizowanie specyfikacji.

Rozważając intuicyjne rozumienie specyfikacji w metodzie aksjomatu przejść, należy znać ogólną postać formuły specyfikacyjnej. Pomijam przy tym część specyfikacji dotyczącą tego, kto wykonuje akcje. Niech $f_1,..., f_n$ będą wewnętrznymi funkcjami stanu, a $g_1,..., g_m$ funkcjami stanu sprzężenia tej specyfikacji.

[4] Choć często myśli się o komunikacji typu CSP jak i o akcji wykonywanej łącznie przez nadawcę i odbiorcę, uznanie jej za akcję kanału eliminuje konieczność wprowadzania pojęcia akcji łącznych.

dokończenie na III okł.

MACIEJ M. SYSŁO
Instytut Informatyki
Uniwersytet Wrocławski

POLSKIE TOWARZYSTWO INFORMATYCZNE

Pierwsza wersja artykułu została przedstawiona podczas Szkoły Jesiennej PTI „Współczesne kierunki rozwoju informatyki" – Mrągowo, 4-8 listopada 1985 r.

Maszyny i algorytmy równoległe (1)

Artykuł ten jest przeglądem modeli maszyn równoległych oraz zaprojektowanych dla nich algorytmów. Zilustrowano podstawowe metody uwspółbieżniania obliczeń: potokowość, rozsyłanie i kumulację oraz najpopularniejsze modele maszyn równoległych: MIMD (tj. wieloprocesory), SIMD z różnymi sieciami połączeń między procesorami oraz maszyny systoliczne. Działanie maszyn równoległych zilustrowano na przykładach pochodzących z dwóch dziedzin: operacji macierzowych i porządkowania. Operacje macierzowe charakteryzują się naturalną współbieżnością tkwiącą w samych problemach; ich algorytmy równoległe są najczęściej odpowiednimi realizacjami klasycznych metod rozwiązywania. Porządkowanie natomiast wymaga całkiem nowych metod realizacji na maszynach równoległych i jest wykonywane za pomocą wąsko wyspecjalizowanych algorytmów zwanych sieciami porządkującymi, złożonych z pewnej liczby prostych modułów porównań.

MASZYNY I ALGORYTMY RÓWNOLEGŁE

Wyróżnić można dwa podstawowe sposoby przyspieszania komputerów: jeden polega na stosowaniu coraz szybszych elementów elektronicznych, a drugi na zastosowaniu współbieżności obliczeń. Pierwszy był siłą napędową rozwoju komputerów od chwili zbudowania pierwszej elektronicznej maszyny cyfrowej ENIAC w 1946 roku, drugi zaś pojawił się dopiero na początku lat sześćdziesiątych, najpierw w rozważaniach teoretycznych, gdy zaczęto interesować się algorytmami równoległymi. Pierwszy ze sposobów przyspieszania obliczeń jest całkowicie uzależniony od dostępnej technologii, drugi natomiast może być rozwijany niezależnie od sprzętu komputerowego. Pomysł uwspółbieżnienia obliczeń pojawił się niemal w tym samym czasie, co konstrukcja pierwszej maszyny. Charlesa Babbage'a (1791–1871), konstruktora pierwszej zachowanej do dzisiaj maszyny liczącej, zainspirowały tablice logarytmiczne, których wykonanie mogłoby pochłonąć jedno całe życie; zostały jednak sporządzone w ciągu kilku lat przez zespół 6 uczonych, 8 wyszkolonych pomocników i około 60 rachmistrzów wykonujących jedynie dodawanie i odejmowanie. Z tego przykładu organizacji obliczeń (współbieżnych, zauważmy!), Babbage zrealizował jedynie ideę maszyny, która miała automatyzować wykonywanie takich samych działań. W późniejszych latach swej działalności zauważył jednak także, że takie obliczenia można by wykonywać za pomocą maszyny, która potrafiłaby produkować wiele wyników w tej samej chwili czasu. Idea ta pojawiła się ponownie dopiero wówczas, gdy rozwój elektroniki przybliżył moment zbudownia pierwszej maszyny równoległej.

Warto wyjaśnić w tym miejscu różnicę między określeniami **równoległy** (ang. *parallel*) i **współbieżny** (ang. *concurrent*). To pierwsze będzie w tym artykule używane przeważnie w odniesieniu do obiektów statycznych, drugie zaś do procesów. Zatem równoległy może być algorytm lub maszyna. Współbieżne są natomiast wykonywane w maszynie procesy lub programy będące implementacjami algorytmów równoległych. Mówimy więc o programowaniu współbieżnym.

Bardzo często będą zamiennie używane określenia maszyna i algorytm. Algorytmy równoległe są bowiem nierozerwalnie związane z modelem (lub konkretną maszyną) obliczeń, dla których zostały opracowane.

Algorytm równoległy można zdefiniować jako zbiór niezależnych zadań, które wykonują się jednocześnie, komunikując się nawzajem, by korzystać ze swoich wyników obliczeń. Klasyfikacja algorytmów równoległych na ogół pokrywa się z klasyfikacją maszyn równoległych, dla których zostały opracowane. Większość algorytmów równoległych można sklasyfikować wg metody sterowania współbieżnością, wielkości rozdrobnienia zadań i geometrii sieci komunikacyjnej.

Sterowanie współbieżnością jest potrzebne do zapewnienia poprawności obliczeń, w których więcej niż jedno zadanie może być wykonywane równocześnie. W tym celu są wymuszane współdziałania między różnymi zadaniami. Najogólniej, obliczenia mogą być synchroniczne dzięki scentralizowanej kontroli, i asynchroniczne, gdy na przykład sterowanie odbywa się za pomocą wspólnych danych.

Stopień rozdrobnienia algorytmu równoległego można określić maksymalną wielkością obliczeń wykonywanych przez zadanie, zanim ma nastąpić komunikacja z innymi zadaniami. Dla przykładu, algorytmy o bardzo dużym rozdrobieniu wymagają zwykle dość częstej komunikacji między zadaniami, a zatem stosunkowo dużą część obliczeń zajmuje komunikacja.

Na podstawie metody sterowania współbieżnością i wielkości rozdrobnienia zadań, wyróżnia się trzy podstawowe typy maszyn równoległych: MIMD, SIMD i maszyny systoliczne.

Maszyna MIMD (ang. *multiple instruction multiple data*), zwana także maszyną wieloprocesorową, składa się z pewnej liczby działających asynchronicznie procesorów, z których każdy ma niezależny licznik rozkazów i wykonuje swój własny program, korzystając przy tym ze wspólnej dla wszystkich procesorów pamięci. Rozdrobnienie zadań w algorytmach dla maszyn MIMD jest zwykle niewielkie i poszczególnym procesorom są przydzielane całe fragmenty obliczeń, np. wywołania procedur. Maszynami są C.mmp, Cm* i Pluribus.

W maszynach SIMD (ang. *single instruction multiple data*) procesor centralny steruje pozostałymi procesorami wysyłając do wszystkich taką samą instrukcję, która jest wykonywana na danych umieszczonych w lokalnych pamięciach procesorów. Działanie maszyn SIMD jest więc synchroniczne. Wielkości zadań obliczeniowych dla poszczególnych procesorów są średnie lub małe. Przykładem maszyny SIMD jest Illiac IV złożona z 64 procesorów.

Gwałtowny rozwój mikroelektroniki w ostatnich latach zrewolucjonizował także konstrukcje komputerów. Technologia VLSI pozwala obecnie umieszczać w jednym układzie (kostce, ang. *chip*) dziesiątki, a nawet setki tysięcy elementów. Doprowadziło to między innymi do powstania całych maszyn (procesorów) w pojedynczych układach, a wśród nich tzw. **maszyn systolicznych**, które mogą być częściami składowymi innych potężniejszych komputerów. Maszyna systoliczna jest regularną siatką elementarnych procesorów o bardzo wąsko wyspecjalizowanym przeznaczeniu, przez które rytmicznie przepływają dane. Jest to więc maszyny synchroniczna ze stałym bardzo dużym rozdrobnieniem zadań.

Jak łatwo zauważyć, procesory (w skrócie **P**) w maszynach MIMD są zwykle pełnymi jednostkami operacyjnymi wyposażo-

nymi we wszystkie podstawowe operacje. W maszynach SIMD rola procesorów jest zwykle ograniczona do wykonywania tylko pewnego podzbioru operacji. W maszynach systolicznych natomiast procesory wykonują najczęściej tylko jedną operację arytmetyczną (i kilka przesłań). Z tego względu procesory w maszynach systolicznych, a czasem także i w maszynach SIMD, są nazywane **elementarnymi procesorami** (ang. *processing element*), w skrócie **PE**.

Sieć połączeń między procesorami w maszynie równoległej może przyjmować różnorodną postać. Bezpośrednie połączenia między procesorami określają komunikację między zadaniami wykonywanymi przez poszczególne procesory. Dla przykładu, procesory mogą być umieszczane w wierzchołkach drzewa binarnego, którego krawędzie wyznaczają możliwe drogi bezpośredniej komunikacji między procesorami. Procesory elementarne mogą tworzyć jedno- lub wielowymiarową sieć, np. liniową, kwadratową lub sześcienną.

Powyższa taksonomia maszyn równoległych, której schemat został zaczerpnięty od Kunga [1], ma swoje odbicie w algorytmach równoległych. Zatem, dla przykładu, algorytm równoległy opracowany dla maszyn SIMD z siecią kwadratową nazywamy algorytmem SIMD z siecią kwadratową itp. Podobnie definiuje się algorytmy systoliczne i MIMD.

W algorytmach systolicznych zadania dla poszczególnych procesorów elementarnych są bardzo proste, a komunikacja między **PE** dość częsta. Algorytmy systoliczne są implementowane zwykle bezpośrednio w postaci odpowiednich układów. Z algorytmami MIMD sytuacja jest niemal odwrotna – są one przeznaczone do wykonywania w systemach wieloprocesorowych o ogólnym przeznaczeniu. Algorytmy SIMD można sklasyfikować między dwoma pozostałymi typami. Na ogół każdy algorytm systoliczny ma odpowiednik w postaci algorytmu SIMD, w którym poszczególne zadania mają mniejsze rozdrobnienia. Nie każdy jednak algorytm SIMD może być zrealizowany przez maszynę systoliczną.

ZŁOŻONOŚĆ I EFEKTYWNOŚĆ ALGORYTMÓW RÓWNOLEGŁYCH

Ocena efektywności algorytmów (i maszyn) równoległych powinna dostarczyć pewnej miary przyspieszenia obliczeń, osiągniętego dzięki użyciu wielu procesorów. W sytuacji idealnej maszyna z **p** procesorami powinna redukować czas obliczeń **p-krotnie**. Dla wielu problemów nie udało się jednak dotychczas tego osiągnąć.

Definicja złożoności czasowej (lub czasu obliczeń) algorytmu równoległego jest naturalnym rozszerzeniem określenia złożoności algorytmu sekwencyjnego. Funkcja $\mathbf{f(n)}$ jest złożonością algorytmu równoległego, jeśli czas od chwili rozpoczęcia pracy przez pierwszy z procesorów maszyny do chwili zakończenia pracy przez wszystkie procesory nie przewyższa $\mathbf{f(n)}$ dla każdego zadania problemu o rozmiarze **n**. Dla przykładu, algorytm sekwencyjny obliczający sumę **n** liczb ma złożoność $\mathbf{0(n)}$, gdyż wykonuje $(\mathbf{n-1})$ dodawań. Jeśli dodawania mogą być wykonywane współbieżnie i dysponujemy $\lceil \mathbf{n/\log n} \rceil$ procesorami, to suma **n** liczb może być obliczona w $\mathbf{2\lceil \log n \rceil}$ krokach za pomocą maszyny drzewiastej. Zatem równoległy algorytm dodawania **n** liczb ma złożoność $\mathbf{0(\log n)}$ przy użyciu $\mathbf{n/\log n}$ procesorów.

Liczba procesorów tworzących maszynę równoległą będzie oznaczana przez **p**, a funkcja złożoności algorytmu równoległego dla takiej maszyny – przez $\mathbf{t_p(n)}$. Zatem $\mathbf{t_1(n)}$ jest złożonością algorytmu sekwencyjnego. Dla wielu maszyn, problemów i algorytmów funkcja $\mathbf{t_p(n)}$ jest określona tylko dla pewnych **p**, a czasem tylko dla jednej wartości **p** zależnej od rozmiaru problemu. Jeśli **p** nie jest ograniczone, to mówi się o **nieograniczonej równoległości** (ang. *unbounded parrallelism*). Iloraz:

$$\mathbf{s_p(n) = t_1(n)/t_p(n)}$$

nazywa się przyspieszeniem (ang. *speedup*) algorytmu równoległego, zaś:

$$\mathbf{e_p(n) = s_p(n)/p}$$

efektywnością (ang. *efficiency*) algorytmu. W definicji przyspieszenia, za $\mathbf{t_1(n)}$ przyjmuje się najczęściej złożoność najlepszego algorytmu sekwencyjnego. Przyspieszenie jest miarą ulepszenia algorytmu sekwencyjnego przez algorytm równoległy, efektywność zaś określa stopień wykorzystania równoległości. Inną miarą efektywności algorytmu równoległego, uwzględniającą czas obliczeń i liczbę procesorów, jest **koszt** (ang. *cost*) algorytmu zdefiniowany jako:

$$\mathbf{c_p(n) = p \cdot t_p(n)}$$

Oczywiste są następujące nierówności:

$$\mathbf{s_p(n) \leqslant p}$$

czyli

$$\mathbf{t_1(n) \leqslant pt_p(n) = c_p(n)}$$

oraz

$$\mathbf{e_p(n) \leqslant 1}$$

gdyż algorytm równoległy o złożoności $\mathbf{t_p(n)}$ może być symulowany przez algorytm sekwencyjny w czasie $\mathbf{0(c_p(n))}$. Warto zauważyć, że $\mathbf{e_1(n) = 1}$, zatem celem niekoniecznie jest optymalizacja $\mathbf{e_p(n)}$ przez wybór odpowiedniej liczby procesorów. Dla wspomnianego wyżej równoległego algorytmu obliczania sumy **n** liczb jest

$$\mathbf{t_p(n) = 2\lceil \log n \rceil,}$$

gdzie $\mathbf{p = \lceil n/\log n \rceil}$, zatem $\mathbf{s_p(n)}$ jest rzędu $\mathbf{p/2}$, a $\mathbf{e_p(n)}$ – w przybliżeniu 1/2, czyli procesory są wykorzystywane przez ten algorytm średnio w 50%.

Algorytm równoległy, dla którego $\mathbf{s_p(n) = p}$, ma optymalny współczynnik przyspieszenia obliczeń (jeśli $\mathbf{s_p(n)}$ jest $\mathbf{0(p)}$, to mówi się o przyspieszeniu optymalnym z dokładnością do stałego współczynnika proporcjonalności). Powodem tego, że wiele algorytmów równoległych nie osiąga optymalnego przyspieszenia, może być np. to, że:

- problem nie daje się podzielić na podproblemy o prawie tych samych rozmiarach,
- korzystanie z zasobów pamięci (zwykle wspólnych) wydłuża pracę procesorów,
- zapewnienie synchronizacji między procesorami pochłania dodatkowy czas.

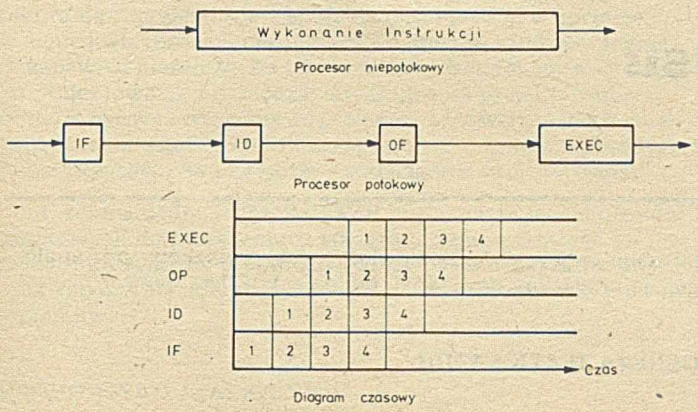
PRZETWARZANIE POTOKOWE

Jednym z najwcześniej zastosowanych podejść do współbieżności obliczeń było **przetwarzanie potokowe** (ang. *pipelining*). Najogólniej mówiąc, potokowość obliczeń jest realizowana przez podział wielokrotnie powtarzanego procesu sekwencyjnego na podprocesy, z których każdy jest wykonywany przez niezależny moduł pracujący równocześnie z innymi modułami.

Jako przykład można rozważyć proces wykonania instrukcji, w którym wyróżnić można przynajmniej cztery podprocesy: pobranie instrukcji **IF**, zdekodowanie operacji **ID**, pobranie argumentów **OF** oraz wykonanie **EXEC**. Na rysunku 1 zilustrowano potokowe przetwarzanie instrukcji. Ten typ potokowości jest realizowany na poziomie systemu operacyjnego. Do niższego poziomu można zaliczyć potokowe jednostki arytmetyczne. Dla przykładu, w procesie dodawania dwóch liczb zmiennoprzecinkowych można wyróżnić następujące podprocesy, które mogą być realizowane potokowo:

- wydzielenie cech (wykładników) obu liczb,
- porównanie cech,
- przesunięcie mantysy jednej z liczb w celu zrównania cech,
- dodawanie mantys,
- normalizacja sumy,

– badanie nadmiaru lub niedomiaru.
Oba typy potokowości zastosowano już w latach sześćdziesiątych w komputerze IBM 360/91, a obecnie są realizowane w większości superkomputerów, np. w TI ASC, CDC 6600, CDC STAR 100, Amdahl 470 V/6 i CRAY 1 (te dwa ostatnie uznaje się za komputery czwartej generacji).



Rys. 1. Potokowe wykonanie instrukcji

Jedną z podstawowych miar oceny działania systemu jest jego współczynnik **przepustowości** (ang. *throughput rate*), definiowany jako liczba wyników (lub wykonanych instrukcji) w jednostce czasu. Potokowość obliczeń jest jedną z metod zwiększania przepustowości systemów. Powróćmy do przykładu z rys. 1. Jeśli czas wykonania instrukcji przez procesor niepotokowy wynosi:

$$\mathbf{T} = t_1 + t_t + t_3 + t_4$$
$$\mathbf{t} = \max\{t_1, t_2, t_3, t_4\}$$

to odpowiednie współczynniki przepustowości wynoszą **1/T** i **1/t**. W szczególności, jeśli wszystkie t_i są sobie równe, to potokowość jest źródłem czterokrotnego zwiększenia przepustowości.

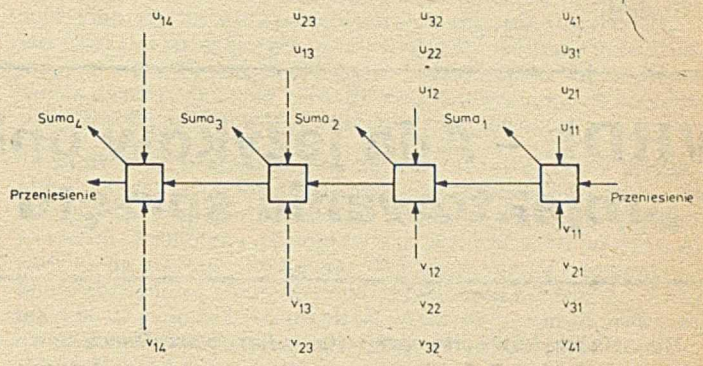
Uzasadnieniem użycia potokowości jest przyjęcie założenia, że ten sam ciąg operacji (instrukcji) będzie powtarzał się bardzo często. Idealną sytuacją dla przetwarzania potokowego są obliczenia związane z wektorami, np. dodawanie dwóch wektorów. Komputery, zdolne w sposób potokowy wykonywać operacje na całych wektorach, nazywa się **komputerami wektorowymi** (ang. *vector computers*). Takimi maszynami są np. STAR 100 i CRAY-1. Dla uproszczenia operacji przesyłania zawartości pamięci, w komputerze STAR 100 zakłada się, że wektor może być utworzony jedynie przez kolejne komórki pamięci, a w maszynie CRAY-1 składowe wektora mogą znajdować się w komórkach pamięci o numerach tworzących ciąg arytmetyczny. Zauważmy, że w drugim wypadku wiersze i kolumny macierzy są wektorami, w pierwszym zaś – tylko jeden typ linii, w zależności od sposobu pamiętania macierzy.

Na rysunku 2 przedstawiono potokowy sumator wektorów o składowych naturalnych. Sumator ten tworzy sumę dwóch wektorów

$(\mathbf{U_1}, \mathbf{U_2}, \mathbf{U_3}, ...)$ i $(\mathbf{V_1}, \mathbf{V_2}, \mathbf{V_3}, ...)$, gdzie:
$\mathbf{U_j} = u_{j1}u_{j2}... u_{jk}$
$\mathbf{V_j} = v_{j1}v_{j2}... v_{jk}$

są binarnymi reprezentacjami składowych.

Czas wykonania operacji wektorowej składa się z dwóch członów: początkowego opóźnienia σ oraz czasu pojawienia się kolejnych wyników τ. Zatem czas wykonania operacji na wektorach długości N wynosi $\sigma + N\tau$. Dla przykładu, w komputerze STAR 100 dla dodawania wektorów mamy $\tau = 0{,}6$ i $\sigma = 71$, a dodawanie dwóch skalarów trwa $\mathbf{t} = 13$ (czasy te są podane w jednostkach cyklu podstawowego, który wynosi 40 nanosekund). Zatem w tym wypadku zastąpienie operacji skalarnych jedną operacją wektorową zrealizowaną potokowo jest opłacalne począwszy już od $\mathbf{N} = 6$ (w ogólnym wypadku, dla $\mathbf{N} > \sigma/(\mathbf{t} - \tau)$. Co więcej, im dłuższe są wektory, tym mniejszy jest wpływ wartości σ na czas obliczeń, gdyż najczęściej $\sigma \gg \mathbf{t}$.



Rys. 2. Potokowy sumator wektorów o składowych naturalnych

Potokowy sumator wektorów z rys. 2 można uznać za maszynę SIMD, a nawet może być zrealizowany przez układ systoliczny. Moim celem jest jednak naszkicowanie ogólnej idei przetwarzania potokowego, które historycznie było pierwszą zrealizowaną koncepcją obliczeń współbieżnych [2]. Zastosowanie potokowości w maszynach SIMD, a zwłaszcza w algorytmach systolicznych, zilustrowano w dalszej części artykułu, poświęconej szczegółowemu omówieniu trzech podstawowych typów maszyn i algorytmów równoległych: MIMD, SIMD i systolicznych. Przykłady problemów i algorytmów pochodzą z dwóch obszarów: obliczeń macierzowych i sortowania. Problemy z pierwszej dziedziny, tak w swoim sformułowaniu, jak i klasycznych algorytmach rozwiązywania, charakteryzują się zwykle jawnie występującą w nich równoległością. Zatem równoległe algorytmy macierzowe są często implementacjami klasycznych algorytmów macierzowych, które zwykle nie są najszybszymi algorytmami sekwencyjnymi. Sortowanie zaś jest tym problemem, dla którego obliczenia współbieżne wymagają metod całkowicie odmiennych od klasycznych, ze względu na niewielkie możliwości uwspółbieżniania tych drugich.

LITERATURA

[1] Kung H. T.: The Structure of Parallel Algorithms. Advances in Computers, Vol. 19, pp. 65-112, 1980
[2] Rammamoorthy C. V., Li H. F.: Pipeline Architecture. Computing Surveys, No. 9, pp. 61-102, 1977.

Efektywność systemu dBase III

dokończenie ze s. 16

Badania wykazały znikomy wpływ rozmiaru relacji wynikowej na efektywność polecenia **JOIN**. Natomiast istotny wpływ na jego efektywność ma typ atrybutu połączeniowego. Jeśli atrybuty połączeniowe są typu numerycznego, to czas wykonywania polecenia **JOIN** przedłuża się blisko trzykrotnie. Jest to związane z przeprowadzaniem każdorazowo konwersji liczb.

* * *

Z przedstawionych wyników widać, że system dBase III powinno się przede wszystkim stosować do zarządzania bazami danych o prostym schemacie. W wypadku baz danych o złożonym schemacie, wymagających stosowania operacji **JOIN**, należy liczyć się ze znacznym obniżeniem efektywności systemu.

LITERATURA

[1] Królikowski Z., Cellary W.: Wstęp do projektowania baz danych - dBase III. WNT, Warszawa, 1988
[2] dBase III. User Manual. Ashton-Tate, 1984.

HENRYK GIZDOŃ
ADAM PAWLAK
WŁODZIMIERZ WRONA
Instytut Elektroniki
Politechnika Śląska
Gliwice

VHDL – Ada języków opisu i projektowania sprzętu

Historia języków opisu sprzętu (ang. *hardware description languages*) [1, 2, 3, 7, 8, 13] ma ponad trzydzieści lat. W tym okresie korzystano z nich najczęściej w środowisku akademickim. Rzadko natomiast używał ich projektant-inżynier, przede wszystkim z powodu ograniczonej zdolności większości języków do reprezentowania problemów praktyki projektowej oraz fragmentarycznego charakteru oferowanych wspomagających narzędzi programowych. Sytuacja ta zmienia się jednak stopniowo wraz z rozwojem technologii wytwarzania układów scalonych. Środowisko projektanta sprzętu musi ulec przeobrażeniu, by mógł on sprostać rosnącej złożoności układów VLSI. Niezbędnym elementem tego środowiska powinien stać się system komputerowo wspomaganego projektanta, wspierający i ewentualnie zastępujący projektanta na różnych etapach procesu projektowania. Jest zupełnie oczywiste, iż w takim nowym dla siebie środowisku projektant stanie wobec podstawowego problemu przekazywania swoich koncepcji projektowych komputerowi.

Język opisu sprzętu może stać się uniwersalnym językiem projektanta, w którym wyrażać on będzie swoje idee projektowe czy struktury konkretnych rozwiązań, i który, porządkując i formalizując koncepcje projektanta, umożliwi komputerowi zweryfikowanie ich poprawności.

Aby język opisu sprzętu mógł zostać uznany za uniwersalne narzędzie projektanta powinien:
- być użyteczny w całym procesie projektowania,
- zawierać naturalne dla projektanta mechanizmy na każdym z jego etapów,
- być językiem ogólnie akceptowanym przez środowisko projektantów sprzętu.

Wymagań tych nie spełnia żaden z ponad stu proponowanych języków. Brak jest ogólnie akceptowanego standardu w tej dziedzinie, a ponadto proponowane języki bardzo rzadko obejmują cały proces projektowania, zmuszając projektanta do korzystania z wielu różnych narzędzi.

Potrzeba opracowania nowego, ogólnie akceptowanego języka opisu sprzętu ujawniła się już w połowie lat siedemdziesiątych. Powołano wówczas międzynarodową grupę roboczą w celu opracowania języka Conlan [10] (ang. *CONsensus LANguage*). Grupa ta opracowała propozycję języka bazowego o nazwie Base Conlan, z którego mogą być formalnie wyprowadzane nowe języki opisu dla specyficznych dziedzin projektowania sprzętu. Zbyt późne (w stosunku do potrzeb środowiska) opracowanie i wydanie raportu Conlanu [9], złożony mechanizm generowania nowych języków oraz brak dostępnych narzędzi programowych wspomagających potencjalnego użytkownika Conlanu spowodowały, iż to interesujące naukowo przedsięwzięcie nie zaowocowało ogólnie akceptowanym standardem.

Nową i – ze względu na sponsora – najpoważniejszą propozycją standardu w dziedzinie języków opisu sprzętu jest język VHDL, który może być uznany Adą języków opisu i projektowania sprzętu[1] (ang. *hardware description and design languages*) [3]. Oba te języki łączy bowiem wiele związków, np. analogiczne przyczyny powstania, budowa, ten sam sponsor.

GENEZA JĘZYKA VHDL

W lipcu 1983 roku zespół złożony z przedstawicieli firmy Intermetrics, IBM oraz Texas Instruments, na zlecenie Departamentu Obrony Stanów Zjednoczonych rozpoczął pierwszy etap pracy, której celem było zaprojektowanie oraz zaimplementowanie standardowego języka opisu i projektowania układów VLSI. Po rocznym okresie projektowania i dwumiesięcznym okresie weryfikacji projektu, przystąpiono do pierwszej implementacji języka. W jej wyniku w grudniu 1985 roku otrzymano pierwszą wersję narzędzia napisanego w języku Ada dla komputerów klasy VA 11/780 i IBM 370.

Projekt języka VHDL jest częścią kompleksowego programu Departamentu Obrony USA o nazwie VHSIC (ang. *very high speed integrated circuits*), którego realizację rozpoczęto w marcu 1980 roku. Jego głównym zadaniem jest opracowanie efektywnych metod projektowania oraz wykorzystania najbardziej złożonych i bardzo szybkich układów scalonych (o szerokości ścieżek $\lambda < 1$ um). W najbliższej przyszłości VHDL stanie się obowiązującym językiem projektowania układów VLSI dla wszystkich kontrahentów Departamentu Obrony.

Zadania języka

VHDL ma spełnić podobną funkcję w dziedzinie projektowania sprzętu, jak język Ada w dziedzinie języków programowania. Jako język standardowy, VHDL ma więc:
- wspierać metodologię systematycznego, hierarchicznego projektowania sprzętu,
- umożliwiać opis projektu oraz jego sprawdzenie w całym procesie jego powstawania (tj. na różnych poziomach abstrakcji),
- umożliwiać gromadzenie doświadczeń projektantów (najlepsze rozwiązania wchodzą w skład biblioteki projektów),
- zapewniać możliwość wykorzystania w projektowanym układzie ostatnich osiągnięć w dziedzinie technologii,
- umożliwiać generowanie nowych wersji projektów realizowanych w nowych technologiach na podstawie rozwiązań projektowych przechowywanych w bibliotece projektów,
- ułatwiać dokumentowanie projektu,
- ułatwiać wymianę informacji między projektantami oraz całymi zespołami projektowymi.

Jednym z głównych wymagań, które postawiono przed projektantami języka VHDL jest zastosowanie Ady zarówno do definicji języka (VHDL adaptuje bez zmian wiele konstrukcji Ady), jak i do jego wspomagających narzędzi (są implementowane w języku Ada).

[1] Jest to coraz częściej stosowany termin omawianej grupy języków komputerowych, akcentujący związek języka z procesem projektowania.

Podstawowe cechy języka

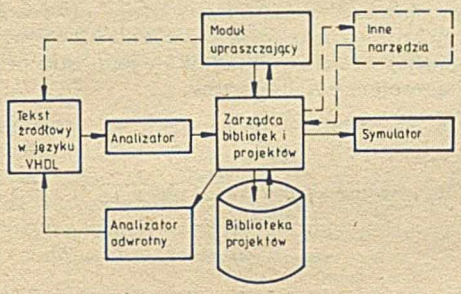
W celu realizacji powyższych zadań zaproponowano język, który ma następujące właściwości:
- wspiera hierarchiczny opis sprzętu,
- obejmuje poziomy abstrakcji: systemowy, architektoniczny, przesłań międzyrejestrowych oraz logiczny,
- umożliwia tworzenie równoważnych modeli funkcjonalnych,
- umożliwia reprezentację dynamiki układu cyfrowego oraz naturalną reprezentację współbieżnych operacji w sprzęcie,
- umożliwia rozdzielenie opisu części sterującej od operacyjnej,
- zapewnia niezależności od konkretnej technologii, metodologii projektowania i narzędzia wspomagania projektowania,
- może być wykorzystany do projektowania w dowolnej technologii, metodologii projektowania lub za pomocą wybranego przez użytkownika narzędzia wspomagania projektowania.

Aktualna definicja języka VHDL nie zawiera mechanizmów językowych do reprezentacji sprzętu układu cyfrowego na poziomie niższym niż bramkowy (tj. na poziomach: układowym oraz topografii układu scalonego) oraz wyższym niż poziom systemowy.

ŚRODOWISKO JĘZYKA VHDL

Dla języka VHDL, jako standardowego w programie VHSIC, opracowano koncepcję otwartego środowiska projektowego, przewidując duże zainteresowanie samym językiem i jego podstawowymi narzędziami wśród projektantów sprzętu, jak i w całym przemyśle narzędzi CAD.

Środowisko podstawowe języka VHDL składa się z sześciu elementów: analizatora (kompilatora), analizatora odwrotnego (rekompilatora), programu zarządzającego biblioteką projektów, modułu upraszczającego, symulatora i biblioteki projektów. Relacje między poszczególnymi elementami przedstwiono na rys. 1. Narzędzia w podstawowym środowisku języka VHDL oraz ewentualne nowe narzędzia dołączone do systemu komunikują się przez dane zawarte w bibliotece projektów. Dostęp do danych w bibliotece jest niezależny od rodzaju komputera.



Rys. 1. Środowisko języka VHDL [6]

Analizator (ang. *analyzer*) tłumaczący postać źródłową w języku VHDL na format pośredni IVAN, weryfikując statycznie (tj. bez prowadzenia symulacji) opisany w języku VHDL model. Format pośredni opiera się na abstrakcyjnym drzewie syntaktyki, konstruowanym podczas analizy leksykalno-syntaktycznej. W kolejnych fazach analizy (kompilacji) drzewo to jest modyfikowane oraz przyporządkowywane mu są dodatkowe atrybuty semantyczne. Podobną metodę zastosowano definiując reprezentację DIANA dla języka Ada.

Zarządca biblioteki projektów (ang. *design library manager*) umożliwia dostęp do biblioteki projektów przechowywanych w formacie pośrednim IVAN.

Analizator odwrotny (ang. *reverse analyzer*) zapewnia rekompilację z formatu pośredniego IVAN na tekst źródłowy w języku VHDL.

Moduł upraszczający (ang. *simplifier*) składa (reorganizuje) hierarchiczną definicję jednostki projektowej równoważną funkcjonalnie poporzedniej strukturze hierarchicznej. W minimalnej konfiguracji systemu może być pominięty.

Symulator weryfikuje poprawność funkcjonalną projektu. Pierwszym etapem działania symulatora jest transformacja opisu na równoważny program w języku Ada, modelujący zachowanie wybranej jednostki sprzętowej. Program ten realizuje symulację na wielu poziomach abstrakcji (na których VHDL jest językiem specyfikacji), umożliwiając jednocześnie symulację jednostek projektowych o różnym rodzaju opisu. Użytkownik ma do dyspozycji liczny zbiór dyrektyw ułatwiających badanie poprawności modelowanego obiektu sprzętowego.

Biblioteka projektów jest zbiorem sprawdzonych projektów, pamiętanych w formacie pośrednim IVAN. Przechowuje się w niej również atrybuty jednostek projektowych oraz wiążących je relacji. Biblioteka projektów ma strukturę hierarchiczną, podzieloną na tzw. konteksty. Projekty będące elementami bibliotecznymi są elementami w określonym kontekście. Kontekst może grupować na przykład elementy pewnego projektu, określonej technologii lub wybranego projektanta.

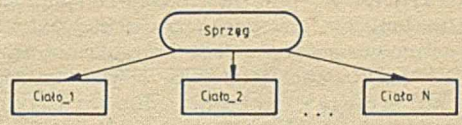
WPROWADZENIE DO JĘZYKA VHDL

Poniżej przedstawiono podstawowe konstrukcje języka, związane z pojęciem tzw. jednostki projektowej, oraz krótko scharakteryzowano pozostałe mechanizmy VHDL. W przedstawionym wprowadzeniu do języka VHDL zrezygnowano z formalizacji i zawężono pojęcia niektórych konstrukcji językowych, w celu zwiększenia przejrzystości i zrozumiałości artykułu dla szerszego grona czytelników.

Jednostka projektowa

Mechanizmem abstrakcji, służącym w języku VHDL do modelowania układów cyfrowych jest jednostka projektowa (ang. *design enitity*). Może ona reprezentować dowolnie złożone elementy lub układy cyfrowe, od prostej bramki logicznej do procesora.

Jednostka projektowa (rys. 2) składa się ze sprzęgu (ang. *interface*) oraz jednego lub kilku alternatywnych ciał (ang. *body*), tworzących razem wzorzec dla określonego fragmentu projektu. Zdefiniowana jednostka projektowa, wchodząca w skład biblioteki projektów, może być elementem bardziej złożonej jednostki projektowej.



Rys. 2. Jednostka projektowa w VHDL

Deklaracja sprzęgu jednostki projektowej

Deklaracja sprzęgu jednostki projektowej udostępnia porty wejściowe i wyjściowe (ang. *I/O ports*) oraz parametry formalne (rodzajowe) innym jednostkom. Pozostałe deklaracje, wspólne dla wszystkich ciał jednostki projektowej, nie są bezpośrednio dostępne dla innych jednostek. Mogą nimi być asercje (ang. *assertions*), które umożliwiają projektantowi sformułowanie ograniczeń dla projektu, a także deklaracje typów, podtypów i atrybutów.

Rozpatrując sumator jednobitowy jako „czarną skrzynkę" można zapisać jego deklarację sprzęgu jak na wydruku 1. SUMATOR jest w poniższym przykładzie nazwą jednostki projektowej sumatora jednobitowego. Porty, parametry formalne i lokalne deklaracje, stanowiące deklarację sprzęgu mogą być użyte we wszystkich ciałach związanych z tym sprzęgiem.

Jednostka SUMATOR ma trzy porty wejściowe (**X, Y, CIN**) i dwa porty wyjściowe (**COUT, SUM**); wszystkie są typu **BIT**. Portowi **CIN** przyporządkowano początkowy stan równy **0**. Słowa zastrzeżone **in** i **out** oznaczają rodzaj (ang. *mode*) portu. W języku VHDL istnieją ponadto trzy inne rodzaje portów: **inout, buffer** i **linkage**. Rodzaj portu determinuje kierunek przepływu informacji przez port, ograniczenia na zapis i odczyt informacji do (z) portu oraz liczbę możliwych źródeł sygnałów. Rodzaje portów **in** i **out** wskazują odpowiednio na przepływ informacji w kierunku i na zewnątrz jednostki projektowej, w której port jest deklarowany. Słowo zastrzeżone **inout** wskazuje natomiast, że informacja jest przekazywana w obu kierunkach. Rodzaj **buffer** oznacza, że port może mieć tylko jedno źródło, rodzaj **linkage** natomiast – że do danego portu można przyłączyć porty o nie określonym kierunku przepływu informacji.

```
entity SUMATOR
  -- bramy wejściowe i wyjściowe
    (X, Y: in BIT;
    CIN : in BIT := '0';
    COUT, SUM : out BIT)  is
  -- parametry formalne
    generic
      (X_COORD : X_INTEGER; Y_COORD : Y_INTEGER)
    -- założenia projektowe dla sumatora
    assert
    ((X_COORD < MAX_COORD) and
    (Y_COORD < MAX_COORD))
      report
      "Błędna wartość współrzędnej położenia sumatora"
end SUMATOR;
```

Wydruk 1. Deklaracja sprzęgu sumatora jednobitowego

Jednostki projektowe mogą mieć różne parametry; ich wartość zależy od otoczenia, w którym jest tworzona nowa konkretna jednostka projektowa. Jednostce projektowej można przyporządkować parametry formalne przez ich wyspecyfikowanie po słowie zastrzeżonym **generic**. W przykładzie jednostki projektowej SUMATOR deklaracja sprzęgu określa klasę sumatorów jednobitnych różniących się parametrami formalnymi **X_COORD** i **Y_COORD**, określającymi współrzędne położenia sumatora, np. w projekcie rozmieszczenia (ang. *placement*) tego układu. Użycie parametrów rodzajowych umożliwia reprezentowanie zależności technologicznych projektu. Założenia weryfikują wartości współrzędnych **X_COORD** i **Y_COORD** położenia sumatora.

Ciało jednostki projektowej

W języku VHDL występują dwa rodzaje ciał jednostki projektowej: architektoniczne (ang. *architectural body*) i konfiguracyjne (ang. *configuration body*). Ciało architektoniczne definiuje strukturę lub zachowanie jednostki. Dopuszcza się również definicję, w której mogą występować jednostki językowe, reprezentujące zarówno strukturę (sprzętową), jak i zachowanie jednostki projektowej.

Ciało konfiguracyjne zawiera informacje o hierarchii składników (ang. *components*) użytych w ciele architektonicznym oraz o aktualnym przyporządkowaniu bram tych składników. W dalszej części artykułu termin ciało architektoniczne zastąpiono krótszym terminem – ciało, a termin ciało konfiguracyjne – terminem konfiguracja ciała (ponieważ zawsze odnosi się do ciała architektonicznego).

W skład ciała jednostki projektowej wchodzą deklaracje obiektów (ang. *objects*; w terminologii języka VHDL są to stałe, zmienne i sygnały), deklaracje składników (ang. *components*) oraz zbiór operujących nimi instrukcji. Deklaracje obiektów konkretyzują (ang. *instantiate*) te obiekty, natomiast konkretyzacja składników jest realizowana za pomocą specjalnej instrukcji konkretyzacji (ang. *instantiation statement*). Instrukcje ciała dzielą się na sekwencyjne i współbieżne.

Instrukcje sekwencyjne mogą przypisywać wartości sygnałom i zmiennym, sterować rozpoczęciem programu oraz kolejnością wykonania jego instrukcji, a także sprawdzać i sygnalizować błędy wykonania tego programu. Mogą one występować wyłącznie w podprogramie (*function, procedure*) lub w instrukcji procesu (*process*). W języku wyróżnia się następujące instrukcje sekwencyjne: zezwolenia, zabronienia, komunikatu (o niespełnieniu założenia), przypisania, wywołania oraz instrukcji sterujących (np. **if, case** itd.).

Instrukcje zezwolenia i zabronienia umożliwiają odsłanianie i zasłanianie zmiennych i sygnałów w opisie zachowania sprzętu. Instrukcja komunikatu służy do definiowania komunikatu dla określonego założenia. Komunikat raportuje sytuację przekroczenia założenia. Instrukcja przypisania nadaje nowe wartości zmiennym oraz aktualizuje stan nośników sygnału (z uwzględnieniem zadeklarowanego opóźnienia). Instrukcja wywołania wywołuje procedurę lub funkcję. Instrukcje sterujące nadzorują kolejność wykonywania innych instrukcji.

```
architecture STRUKTURA of SUMATOR is
  BLOK_STRUKTURALNY:
    block
      -- deklaracja sygnałów wewnętrznych
      signal S1, S2, S3, S4: BIT ;
      -- lokalne deklaracje bramek sumatora
      component XOR_GATE port (A, B: in BIT; C: out BIT);
      component AND_GATE port (A, B: in BIT; C: out BIT);
      component OR_GATE  port (A, B: in BIT; C: out BIT);
    begin
     -- struktura sumatora - konkretyzacja jego składników
      X1 : XOR_GATE port (X, Y, S1) ;
      X2 : XOR_GATE port (S1, CIN, SUM) ;
      A1 : AND_GATE port (CIN, S4, S2) ;
      A2 : AND_GATE port (X, Y, S3) ;
      OR1 : OR_GATE port (S2, S3, COUT) ;
      OR2 : OR_GATE port (X, Y, S4) ;
    end block ;
end   STRUKTURA;
```

Wydruk 2. Deklaracja ciała sumatora – opis strukturalny

Naturalnym mechanizmem opisu sprzętu są instrukcje współbieżne, takie jak: blok, proces, współbieżne przypisanie sygnału, konkretyzacja oraz powielenie. Instrukcja bloku jest konstrukcją zawierającą lokalne deklaracje i instrukcje współbieżne, składające się na ciało jednostki projektowej. Instrukcja procesu natomiast zawiera instrukcje sekwencyjne, które obliczają aktualne wartości zmiennych i sygnałów. Instrukcja konkretyzacji konkretyzuje składnik z zadeklarowanego wzorca wybranej jednostki projektowej. Instrukcja powielania umożliwia konkretyzację regularnej struktury składników jednostki projektowej wraz z przyłączeniem ich do portów.

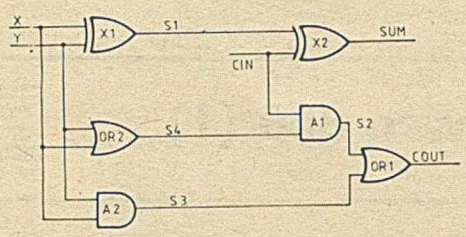
```
architecture ZACHOWANIE of SUMATOR is
  BLOK_FUNKCJONALNY:
    block
      signal A : BIT ;
    begin
      SUM <= X xor Y xor CIN after 4 ns ;
      A <= (Y and CIN) or (X and CIN) or (X and Y) ;
      COUT <= A after 5 ns ;
    end block ;
end ZACHOWANIE ;
```

Wydruk 3. Deklaracja ciała sumatora – opis funkcjonalny

Na przykładzie sumatora, którego sprzęg opisano na wydruku 1, omówimy alternatywne deklaracje jego ciał, specyfikujących zarówno jego strukturę, jak i zachowanie (wydruk 2 i 3). Schemat logiczny struktury sumatora, opisanej na wydruku 2, przedstawiono na rys. 3.

W przykładzie tym konkretyzowane są składniki jednostek projektowych **XOR_GATE, AND_GATE** i **OR_GATE** o nazwach **X1, X2, A1, A2, OR1**. Połączenia między utworzonymi składnikami są realizowane przez użycie nośników **S1, S2, S3, S4**. Nośnik sygnału jest charakterystycznym pojęciem języka opisu sprzętu i umożliwia przekazywanie zmian wartości (historii) sygnału (model asynchroniczny).

W typowej sytuacji sygnał przekazuje wartość bez jej zapamiętania. Jeżeli projektant chce zachować przekazywaną wartość sygnału (modelując zachowanie układu synchronicznego) to musi poprzedzić taki sygnał słowem zastrzeżonym **memoried**.



Rys. 3. Schemat struktury bramkowej sumatora jednobitowego

Ciało sumatora z wydruku 3 stanowi jego model funkcjonalny, uwzględniający czasy opóźnień. Zależności wyjściowo-wejściowe w modelu funkcjonalnym są reprezentowane przez zbiór instrukcji przypisania sygnału. Przypisanie nowej wartości sygnału może być indywidualnie opóźniane.

Elementy języka VHDL

Deklaracja typów. Język VHDL zawiera kilka standardowych typów i podtypów. Projektant ma ponadto możliwość deklarowania nowych typów. Typy definiowane przez użytkownika można podzielić na skalarne i złożone. Typy skalarne dzielą się na typy wyliczeniowe, numeryczne i fizykalne. Typ fizykalny umożliwia wyrażanie wielkości będących jednostkami miary. Typy złożone składają się z typów skalarnych lub innych typów złożonych. Wyróżniono dwie klasy typów złożonych: tablice i rekordy.

Klasy obiektów. W języku wyróżnia się trzy klasy obiektów: stałe, zmienne i sygnały. Wszystkie obiekty są tworzone przez ich deklarację. Deklaracja specyfikuje nazwę obiektu, klasę obiektu i typ danych obiektów. Niektóre deklaracje obiektów specyfikują dodatkową informację: deklaracja stałej określa wartość tej stałej; deklaracja zmiennej może specyfikować początkową wartość zmiennej; deklaracja sygnału wielobitowego (magistrali) określa funkcję arbitra (ang. *bus resolution function*).

Podprogramy. W języku VHDL występują dwa rodzaje podprogramów: funkcje i procedury. Funkcje służą przede wszystkim do obliczeń. Dodatkowym zadaniem funkcji jest wyznaczanie wartości sygnałów wspólnej magistrali. Procedura może być używana jako mechanizm dekompozycji złożonego zachowania jednostki projektowej.

Pakiety. Pakiet jest mechanizmem grupującym deklaracje. Przez umieszczenie deklaracji wewnątrz pakietów zawarte w pakiecie dane projektowe można podzielić między kilku projektantów. Za pomocą pakietów magazynuje się deklaracje w bibliotece w celu późniejszego ich wykorzystania. Pakiety stosuje się również do zarządzania tekstem oraz do tworzenia wyodrębnionych modułów projektowych. W pakietach grupuje się deklaracje typów, podtypów, atrybutów, stałych, podprogramów i składników stanowiących deklaracje wybranych jednostek projektowych. Raz zdefiniowany pakiet może być użyty w wielu projektach.

* * *

Zalet języka VHDL należy upatrywać w następujących mechanizmach i cechach:

- alternatywnym opisie jednostki projektowej dla podanego explicite jej sprzęgu,
- złożonym modelu dynamiki uwzględniającym opóźnienia transmisji sygnału przez układ, zarówno intercyjne, jak i transportowe,
- konwersji typów sterowanej przez użytkownika,
- niezależności od metodologii projektowania i technologii,
- wykorzystaniu podstawowego pojęcia języka Ada, tzw. pakietu,
- akceptowaniu mieszanego opisu złożonego z operacji współbieżnych i sekwencyjnych,
- dopuszczeniu mieszanego opisu zachowania i struktury jednostki,
- ogólnym mechanizmie parametryzacji jednostki projektowej.

Język VHDL jest obecnie poddawany procesowi normalizacji w jednym z podkomitetów IEEE. Oczekuje się, iż język oraz jego otwarte środowisko projektowe staną się dynamicznym nośnikiem nowych metod projektowania układów VLSI.

LITERATURA

[1] Barbacci M.: Structural and behavioural description of digital systems. J. Tiberghien (Ed.), New computer architectures, Academic Press, 1984
[2] Computer: Vol. 7, No. 12, 1974; Vol. 10, No. 6, 1977; Vol. 18, No. 2, 1985 (specjalne wydania poświęcone językom opisu sprzętu)
]3] Dasgupta S.: Computer Design and Description Languages. Advances in Computers, Vol. 21, pp. 19-154, 1982
[4] Devey A.: VHSIC Hardware Description (VHDL) Development Program. Proc. 20th Design Automation Conference, 1983
[5] Devey A.: VHSIC Hardware Description (VHDL) Language Program. Proc. 21st Design Automation Conference, 1984
[6] IEEE Design and Test, April 1986 (specjalne wydanie poświęcone językowi VHDL)
[7] Marczyński R., Bąkowski P.: What Do the Computer Hardware Description Languages Describe? Proc. Int. Symp. on Computer Hardware Description Languages and their Applications, Palo Alto, 1979
[8] Pawlak A.: A Turorial Guide to Modern Hardware Description and Design Languages. Waldschmidt K. and Myrhraug B. (Eds.), Microocomputers, usage and design, Elsevier Science Publishers, 1985
[9] Piloty R. et al.: CONLAN Report. Springer-Verlag, 1983
[10] Piloty R., Borrione D.: The CONLAN Project – Concepts, Implementations and Applications, Computer, February 1985
[11] Shahdad M., Ackley D., Brown M.: VHSIC Hardware Description Language Overview, Proc. of ACM Annual Conference, 1934
[12] Shahdad M. et al.: VHSIC Hardware Desription Language. Computer, February 1985
[13] Shiva S.: Computer Hardware Description Languages – A Tutorial. Proc. of the IEEE, Vol. 67, no. 12, December 1979
[14] Vaxman R.: Hardware Design Languages for Computer Design and Test. Computer, April 1986
[15] Zalewski J.: Ada – nowy język programowania. Informatyka, nr 11-12, 1981; nr 1-6, 1982.

CeBIT '89

Hanowerski CeBIT jest od kilku lat bezspornie największą na świecie prezentacją aktualnych osiągnięć i tendencji rozwojowych technologii informatycznej. Prymat ten dotyczy zarówno liczby wystawców i powierzchni ekspozycji, jak i liczby zwiedzających. Daleko w tyle pozostał pierwowzór imprezy, jakim był niewątpliwie paryski SICOB, a także amerykański COMDEX, który niegdyś szokował rozmiarami ze względu na dominującą pozycję USA i kontynentalne rozmiary tego rynku.

CeBIT '89, który odbędzie się w dniach od 8 do 15 marca, potwierdzi kolejny raz rosnącą przewagę nad imprezami konkurencyjnymi. Sprawi to niewiarygodna liczba ok. 3000 wystawców z 35 krajów oraz zarezerwowane dla nich 325000 m² powierzchni. Prognozy oparte na wieloletnich obserwacjach frekwencji przewidują ponad 500000 zwiedzających (w 1988 r. 485800).

Program CeBIT '89 przewiduje 12 podstawowych obszarów tematycznych, zlokalizowanych w oddzielnych halach lub ich wydzielonych częściach:

- telekomunikacja,
- systemy komputerowe i automatyzacja biur,
- urządzenia peryferyjne,
- mikrokomputery,
- zastosowanie w małych przedsiębiorstwach i wolnych zawodach,
- oprogramowanie, banki danych i doradztwo,
- techniki komputerowego wspomagania projektowania i wytwarzania,
- technologia rejestracji danych,
- zastosowania w bankowości i działalności finansowej,
- techniki zabezpieczeń,
- badania i rozwój technologiczny,
- szkolenie i doskonalenie kadr.

Innym elementem zwiększania atrakcyjności CeBIT będzie kilkanaście ekspozycji specjalistycznych, wśród których należy m.in. wymienić adresowany do młodzieży „Computer Camp" oraz ekspozycję osiągnięć sztuki komputerowej „Artware".

WOJCIECH CELLARY
JERZY KRĘGLEWSKI
Instytut Automatyki
Politechnika Poznańska

RUTA MAĆKOWIAK
Instytut Komputerowych Systemów
Automatyki i Pomiarów
Wrocław

System mikrokomputerowy Elwro 800 (1)

Jednym z najczęściej zadawanych pytań dotyczących systemu mikrokomputerowego Elwro 800 jest pytanie, czy jest on mniej lub bardziej wierną kopią mikrokomputera osobistego IBM PC/XT, czy też zupełnie innym, niezależnym rozwiązaniem. Celem niniejszego artykułu jest odpowiedź na to i inne pytania dotyczące Elwro 800, przez przedstawienie i uzasadnienie przyjętych założeń koncepcyjnych tego systemu.

Firma IBM weszła późno na światowy rynek mikrokomputerowy. Przeskoczyła epokę mikrokomputerów ośmiobitowych, pominęła obszar mikrokomputerów domowych i rynek mikrokomputerów szesnastobitowych, przeznaczonych do zastosowań profesjonalnych.

Od momentu ustabilizowania się firmy IBM na rynku mikrokomputerów osobistych obserwujemy dynamiczny rozwój tzw. mikrokomputerów kompatybilnych z IBM PC. Istnieją dwie klasy tych mikrokomputerów: klasa mikrokomputerów o obniżonych parametrach funkcjonalnych, niezawodnościowych, efektywnościowych oraz klasa mikrokomputerów o podwyższonych parametrach w stosunku do oryginalnego IBM PC. Pierwsza z wymienionych klas jest obecnie domeną producentów azjatyckich (Tajwan, Hongkong, Korea Płd.), druga natomiast – wiodących firm japońskich, amerykańskich i europejskich, np. NEC, Wang, Olivetti. W obu klasach istotą kompatybilności jest zgodność programowa, natomiast sprzętowo poszczególne rozwiązania odbiegają, i to nawet dość znacznie, od rozwiązań oryginalnego IBM PC.

Przedstawiona analiza dotyczy profesjonalnych mikrokomputerów osobistych. Na zjawisko IBM PC należy jednak spojrzeć inaczej, w kontekście całego zakresu zastosowań mikrokomputerów, który można ogólnie podzielić na dwie części: w pierwszej mieszczą się zastosowania nazywane umownie personalnymi (obliczeniowymi), a w drugiej – przemysłowymi. Zastosowania personalne to takie, w których zarówno wejście, jak i wyjścia systemu mikrokomputerowego są ukierunkowane na komunikację z człowiekiem. Zastosowania przemysłowe to takie, w których co najmniej część wejść lub wyjść systemu mikrokomputerowego służy do komunikacji z szeroko rozumianym procesem przemysłowym (w automatyce, robotyce, telekomunikacji itp.). Przy tak postawionym zagadnieniu widać wyraźnie, że obszar zastosowań mikrokomputera IBM PC ograniczony jest do środkowej części obszaru zastosowań personalnych. Nie obejmuje on prostych zastosowań personalnych, do których wystarczają mikrokomputery ośmiobitowe, gdyż jest za drogi. Nie obejmuje również najbardziej złożonych zastosowań personalnych mikrokomputerów, gdzie wymagane są systemy wielomikroprocesorowe, o megabajtowych pojemnościach pamięci operacyjnej. Nie obejmuje też oczywiście obszaru zastosowań przemysłowych.

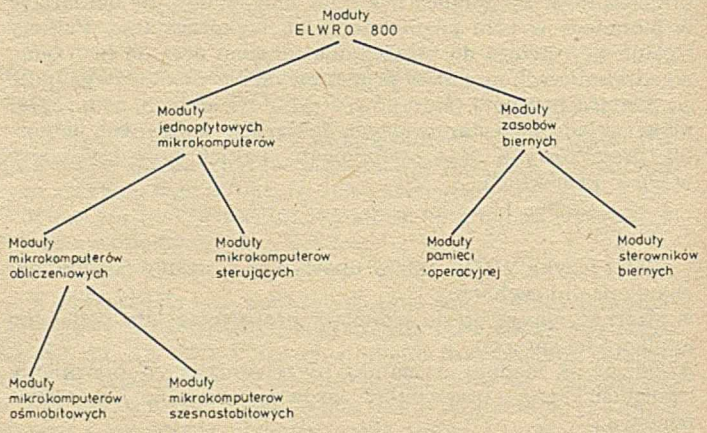
Reasumując należy stwierdzić, że mikrokomputer IBM PC/XT cechuje się zamkniętą, uproszczoną, w stosunku do możliwości mikroprocesorów szesnastobitowych, strukturą sprzętową i bardzo bogatym oprogramowaniem do zastosowań personalnych. Jego główna wartość leży więc w oprogramowaniu. Drugorzędne znaczenie mają natomiast jego rozwiązania sprzętowe, w których nie ma wieloprocesorowości, pamięć operacyjna RAM jest ograniczona do 640 KB i bardzo silnie ograniczone są możliwości rozbudowy konfiguracji podstawowej.

ARCHITEKTURA SYSTEMU

System mikrokomputerowy Elwro 800 został opracowany z myślą o zaspokojeniu specyficznych wymagań polskiego rynku. Punktem wyjścia było dążenie do skonstruowania systemu o możliwie jak najszerszym obszarze zastosowań, obejmującym zarówno zastosowania personalne, jak i przemysłowe. Oczywiście nie jest możliwe skonstruowanie jednego uniwersalnego mikrokomputera, który spełniałby wszystkie wymagania jednego i drugiego działu zastosowań. Dlatego Elwro 800 zaprojektowano jako rodzinę systemów mikrokomputerowych.

System Elwro 800 ma budowę modułową. W skład Elwro 800 (rys. 1.) wchodzą dwa podstawowe rodzaje modułów: jednopłytowe mikrokomputery oraz moduły zasobów biernych.

Obecnie są dostępne dwa moduły jednopłytowych mikrokomputerów obliczeniowych: pierwszy, szesnastobitowy z mikroprocesorem Intel 8086 i koprocesorem arytmetycznym Intel 8087 oraz drugi ośmiobitowy z mikroprocesorem Intel 8080. Oba te moduły składają się z następujących bloków funkcjonalnych:



Rys. 1. Klasyfikacja modułów ELWRO 800

Dr inż. RUTA MAĆKOWIAK ukończyła w 1957 r. studia na Wydziale Łączności Politechniki Wrocławskiej. Pracę zawodową zaczęła we Wrocławskim Oddziale Instytutu Łączności. Od 1973 r. pracuje w Instytucie Komputerowych Systemów Automatyki i Pomiarów we Wrocławiu, gdzie od 1981 r. kieruje Ośrodkiem Systemów Obiektowych, w którym jest opracowywany i przygotowywany do wdrożenia do produkcji system mikrokomputerowy ELWRO-800.

pamięci operacyjnej RAM (128 KB – mikrokomputer szesnastobitowy, 64 KB – mikrokomputer ośmiobitowy), pamięci operacyjnej EPROM (odpowiednio 384 KB i 32 KB), szeregowego portu we-wy Intel 8251A, równoległego portu we-wy Intel 8255, programowalnych liczników Intel 8253, sterownika przerwań Intel 8259A oraz układu sterowania i arbitrażu wielomikroprocesorowej magistrali systemowej IEC-AMS. Pamięć operacyjna RAM jest pamięcią podwójnie adresowaną (ang. *dual-port RAM*).

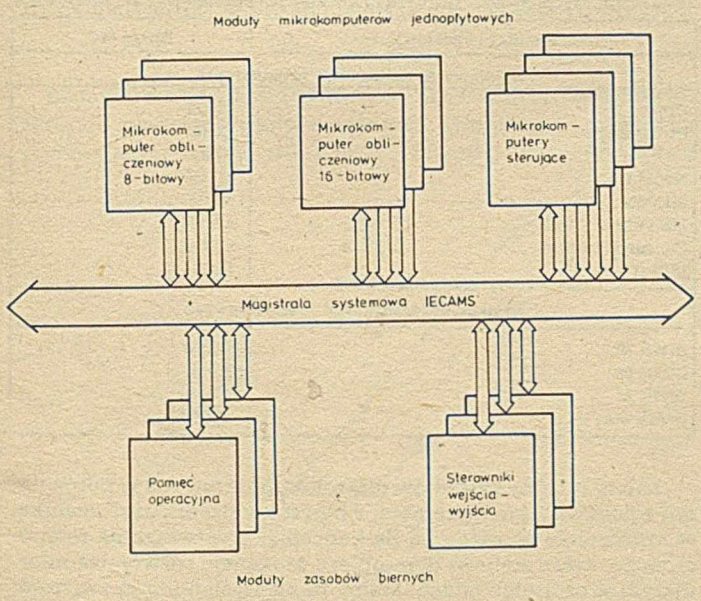
Moduły mikrokomputerów sterujących są inteligentnymi sterownikami urządzeń zewnętrznych, takich jak: dyski elastyczne, dysk stały Winchester, grupa monitorów znakowych, sieć lokalna itp. Każdy moduł wyposażony jest w mikroprocesor, który umożliwia interpretację złożonych poleceń z mikrokomputerów obliczeniowych i wstępne przetwarzanie.

Moduły pamięci umożliwiają rozszerzenie pojemności pamięci prywatnej umieszczonej na płytach modułów mikrokomputerów obliczeniowych. Maksymalna pojemność pamięci operacyjnej jednopłytowych mikrokomputerów szesnastobitowych została rozszerzona z 1 MB do 16 MB, natomiast mikrokomputerów ośmiobitowych – z 64 KB do 1 MB. Dotychczas opracowano dwa moduły pamięci dla Elwro 800: pierwszy o pojemności 0,5 MB z kontrolą parzystości oraz drugi o pojemności 256 KB z automatyczną korekcją błędów (ECC).

Moduły sterowników biernych różnią się od modułów mikrokomputerów sterujących tym, że są widziane przez mikrokomputery obliczeniowe jako fragmenty pamięci operacyjnej lub urządzenia zewnętrzne. Opracowano dwa takie moduły, a mianowicie moduł grafiki kolorowej oraz moduł sterowania telewizorem, stosowanym jako monitor ekranowy.

Ważną cechą architektury Elwro 800 jest wielomagistralowość. Podstawową magistralą jest wieloprocesorowa magistrala systemowa, przez którą komunikują się między sobą wszystkie moduły tworzące konkretną konfigurację systemu. Do tej magistrali dołączone są zarówno moduły mikrokomputerów obliczeniowych i sterujących, jak i moduły zasobów biernych. Każdy z modułów mikrokomputerów zawiera niezbędne układy arbitrażu magistrali systemowej, umożliwiające współpracę na niej od jednego do szesnastu mikrokomputerów.

Poszczególne moduły mikrokomputerów mają swoje wewnętrzne magistrale, które służą do komunikacji z pamięcią i układami wejścia-wyjścia umieszczonymi w danym module. Wydzielenie w systemie wielu magistrali zwiększa jego efektywność oraz podnosi niezawodność działania systemu jako całość. Zwiększenie efektywności uzyskuje się dzięki temu, że w trakcie wykonywania większości operacji poszczególne mikroprocesory korzystają z rozłącznych magistral i tym samym unikają konfliktów powstających przy współubieganiu się o magistralę systemową. Wzrost niezawodności wynika z tego, że dzięki fizycznemu rozdzieleniu magistral oraz sprzętowym zabezpieczeniom przed dostępem mikroprocesora jednego modułu do wewnętrznej magistrali drugiego – błędne działanie jednego z modułów nie powoduje zakłócenia pracy pozostałych. Ponadto uszkodzenie jednego z modułów może być wykryte przez pozostałe moduły mikrokomputerów i funkcje dotąd realizowane w module, który uległ uszkodzeniu, mogą być całkowicie lub częściowo przejęte przez pozostałe moduły systemu.



Rys. 2. Architektura systemu ELWRO 800

Architektura systemu została pokazana na rys. 2, a opis poszczególnych modułów podano w drugiej części artykułu.

OPROGRAMOWANIE SYSTEMOWE

W skład oprogramowania systemowego Elwro 800 wchodzi dwanaście różnych operacyjnych będących funkcjonalnymi odpowiednikami systemów uznanych za standardy światowe. W tabeli zestawiono nazwy systemów operacyjnych stosowanych na świecie oraz ich funkcjonalnych odpowiedników przygotowanych dla Elwro 800. Systemy te można podzielić na dwie grupy. W grupie pierwszej znajdują się systemy przeznaczone do zastosowań personalnych, takich jak: automatyzacja prac biurowych, wspomaganie zarządzania, wspomaganie projektowania, nauczanie, obliczenia inżynierskie, itp. W drugiej grupie znajdują się wielozadaniowe systemy operacyjne czasu rzeczywistego przeznaczone do zastosowań przemysłowych w układach automatyki, robotach, telekomunikacji itp.

Ponieważ charakterystyki tych systemów są dość powszechnie znane, to pominiemy w tym artykule ich szczegółowy opis. Wspomnimy jedynie o systemie operacyjnym CP816, który został opracowany specjalnie dla mikrokomputera Elwro 800. Jest to system jednozadaniowy, dwuprocesorowy, kompatybilny z dwoma systemami CP/M 2.2 i CP/M 86. Przeznaczony jest dla konfiguracji Elwro 800 zawierającej moduły mikrokomputerów obliczeniowych ośmio- i szesnastobitowych. System CP816 pracując na obu procesorach komunikuje się z operatorem przez jedną konsolę operatorską i umożliwia eksploatowanie całego oprogramowania przygotowanego dla systemów CP/M i CP/M 86. Programy zlecone przez operatora do wykonania są rozpoznawane przez system i kierowane do odpowiedniego modułu.

System CP816 jest więc dużym ułatwieniem w pracy użytkowników, którzy pragną wykorzystać oprogramowanie zarówno z mikrokomputerów ośmiobitowych, jak i szesnastobitowych. Takie potrzeby pojawiają się często, gdy użytkownicy rozpoczynają dopiero wykorzystywanie mikrokomputera szesnastobitowego w swojej pracy, mając za sobą określony dorobek na mikrokomputerze ośmiobitowym. Wówczas zgromadzone oprogramowanie mikrokomputera ośmiobitowego może być wykorzystywane równolegle z wprowadzeniem oprogramowania dla mikrokomputera szesnastobitowego.

RODZINA SYSTEMÓW

Elwro 800 jest rodziną systemów mikrokomputerowych. Z modułów Elwro 800 można złożyć wiele istotnie różniących się między sobą konkretnych systemów, które można dowolnie rozbudowywać.

Najprostszą konfigurację systemów tworzy pojedynczy moduł mikrokomputera obliczeniowego ośmio– lub szesnastobitowego. Moduł taki jest funkcjonalnie pełnym mikrokomputerem i może być stosowany jako sterownik mikroprocesorowy w zastosowaniach przemysłowych. Przez dobór typu mikroprocesora, nieobsadzanie układami bloków funkcjonalnych nadmiarowych w danym zastosowaniu oraz stosowanie układów pamięciowych o pojemnościach mniejszych niż maksymalne można dostosowywać strukturę i moc obliczeniową mikrokomputera do wymagań konkretnego zastosowania.

Drugą podstawową konfiguracją Elwro 800 jest system jednoprocesorowy, wielomodułowy. Przez jednoprocesorowość należy rozumieć jeden moduł mikrokomputera obliczeniowego ośmio- lub szesnastobitowego. Inne moduły wchodzące w skład systemu w tej konfiguracji to moduły mikrokomputerów sterujących, pamięci operacyjnej oraz sterowników biernych. W tej klasie systemów można oczywiście uzyskać bardzo wiele konkretnych systemów dołączając różne moduły do mikrokomputera obliczeniowego. Naturalne jest też dołączanie modułów specjalnie skonstruowanych dla danego zastosowania.

Trzecią podstawową konfiguracją Elwro 800 jest homogeniczny system wieloprocesorowy, a więc system złożony z wielu modułów mikrokomputerów obliczeniowych, albo ośmio- albo szesnastobitowych i modułów mikrokomputerów sterujących, pamięci operacyjnej i sterowników biernych. Liczba mikrokomputerów wchodzących w skład takiego systemu jest ograniczona do szesnastu.

Ostatnią, najbardziej złożoną konfiguracją Elwro 800 jest heterogeniczny system wieloprocesorowy, w którym dopuszcza się współdziałanie mikrokomputerów obliczeniowych ośmio- i szesnastobitowych.

Moduły Elwro 800 są zrealizowane w postaci pakietów w formacie podwójnym Eurocard o wymiarach 233,4 mm na 220 mm. Każdy moduł ma dwa złącza pośrednie 96-stykowe typu Eltra 821. Złącza te są wykorzystywane do połączeń pakietu z wielomikroprocesorową magistralą systemową. Zgodnie ze standardem Eurocard, stroną montażu elementów na pakiecie jest strona prawa, patrząc od strony wkładania pakietów do kaset. Urządzenia zewnętrzne są dołączane do pakietów za pomocą złączy szufladowych Eltra. Większość pakietów jest wykonana na obwodach drukowanych czterowarstwowych, przy czym warstwy wewnętrzne są wykorzystywane jedynie do rozprowadzenia zasilania.

Podstawowym elementem konstrukcyjnym systemu jest 19-calowa kaseta o wysokości **6U** dostosowana do pakietów Eurocard, mająca 21 pól pakietowych (stanowisk). Obecnie stosowane są dwa wykonania kasety Elwro 800: kaseta z 12 polami pakietowymi oraz kaseta z 6 polami pakietowymi. W pierwszej kasecie pozostałe 9 pól pakietowych zajmują 3 moduły zasilaczy. Druga kaseta, węższa niż 19 cali, obok 6 pól pakietowych zawiera dwa moduły zasilaczy, które zajmują następne 6 pól. Połączenia między złączami pól pakietowych są wykonane na platerze drukowanym 4-warstwowym.

Dla Elwro 800 opracowano typoszereg zasilaczy impulsowych ZIS-3. Jeden zasilacz ZIS-3 zajmuje w kasecie 3 pola pakietowe. Zasilacze ZIS-3 wykonywane są w trzech typach:
ZIS-3-11 (+ 5V/16A, − 5V/2A)
ZIS-3-12 (+ 12V/6A, − 12V/2A)
ZIS-3-04 (+ 24V/4A)
W konstrukcji zasilaczy przewidziano stosowanie zasilania buforowanego.

System Elwro 800 jest wdrażany do wielkoseryjnej produkcji. Do czasu wydrukowania tego artykułu powinien ukazać się w jednej konfiguracji – Elwro 826, z kasetą na 12 pakietów, z zasilaczami ZIS-3-12 i dwoma ZIS-3-11. Druga kaseta zawiera 2 napędy dysków elastycznych 5,25 cala. Podstawowe moduły to M-M16, M-M08, M-RAM, M-FLO, M-UKZ (por. w następnej części artykułu), a urządzenia zewnętrzne to drukarka D-100 lub jej odpowiednik, monitor ekranowy MERA 7953N lub jego odpowiednik. Jest to konfiguracja, która umożliwia użytkownikowi rozbudowę sprzętu i oprogramowania według jego potrzeb, w miarę wdrażania do produkcji nowych modułów.

ZAKRES ZASTOSOWAŃ

Na tle krótkiej charakterystyki systemu mikrokomputerowego Elwro 800 można przedstawić jego zalety z punktu widzenia użytkownika oraz producenta. Jeśli przez użytkownika systemu rozumie się nie osobę bezpośrednio wykorzystującą mikrokomputer, lecz obiekt przemysłowy, gospodarczy, administracyjny itp., który należy skomputeryzować (np. ciąg produkcyjny, grupę robotów, pocztę, oddział banku, szpital itp.), to u tak rozumianego użytkownika występuje zawsze hierarchia potrzeb. Rozumiemy przez to zapotrzebowanie na mikrokomputery o różnej mocy obliczeniowej. Przykładowo, w banku potrzebny jest prosty system do obsługi okienek, być może system wielodostępny ze względu na małe obciążenie, większy system dla kontroli i ewidencji, inny system dla komunikacji z dużym komputerem itp. W robotyce potrzeba stosunkowo małych mikrokomputerów do sterowania poszczególnymi ruchami ramion, dużego mikrokomputera do rozpoznawania obrazów z kamery, specjalnego mikrokomputera do komunikacji z innymi robotami i operatorem itp. oraz niezależnego systemu uruchomieniowego do przygotowywania oprogramowania.

Jeśli tak rozumiany obiekt, w którym występuje cała hierarchia potrzeb, próbować komputeryzować przy wykorzystaniu mikrokomputerów różnych typów i różnych firm, to szanse zbudowania zintegrowanego systemu zastosowaniowego są dość małe, a nakłady i wysiłek duże, przede wszystkim ze względu na naturalną niekompatybilność sprzętową i programową różnych mikrokomputerów. Budowa takiego zintegrowanego systemu zastosowaniowego na podstawie jednej rodziny systemów mikrokomputerowych, jaką jest Elwro 800, daje natomiast natychmiastowe korzyści: jedność sprzętu, jedność oprogramowania, dobrze rozwinięte, jednolite środki uruchomieniowe, jednolitość serwisu, dokumentacji i szkolenia obsługi. Trzy pierwsze cechy mają podstawowe znaczenie na etapie konstruowania systemu zastosowaniowego, trzy następne – na etapie eksploatacji.

Budowanie systemu zastosowaniowego opartego na rodzinie systemów mikrokomputerowych ma również inną ważną zaletę. Jest nią możliwość zaspokajania potrzeb zmiennych w czasie, co wynika przede wszystkim z naturalnej konieczności rozwoju systemu. Jeśli zastosowany mikrokomputer ośmiobitowy przestanie wystarczać ze względu na wzrost obciążenia po jakimś czasie eksploatacji, to stosując Elwro 800 mamy możliwość dołączenia drugiego mikrokomputera ośmiobitowego i podziału obciążenie, zastąpienie mikrokomputera ośmiobitowego przez komputer szesnastobitowy, dołożenia mikrokomputera szesnastobitowego i rozwijania systemu na tym mikrokomputerze przy pozostawieniu dotychczasowych funkcji w mikrokomputerze ośmiobitowym. Jest również możliwość zmiany typów i liczby urządzeń zewnętrznych, np. z dysków elastycznych pięciocalowych na ośmiocalowe lub na dyski stałe i zwiększania pojemności pamięci operacyjnej. Wszystkich tych zmian można dokonywać bez burzenia dotychczasowej koncepcji systemu zastosowaniowego.

Zestawienie systemów operacyjnych Elwro 800 i ich odpowiedników światowych

Standard światowy	Elwro 800
Systemy do zastosowań personalnych	
CP/M 2.2	CPO8
CP/M-86	CP16
—	CP816
MP/M II	MPO8
MP/M86	MP16
MS-DOS (PC-DOS)	MS16
Concurrent CP/M	CCP16
ISIS-II	ISO8
Systemy do zastosowań przemysłowych	
iRMX 80	RXO8
iRMX 88	SX16
iRMX 86	RX16
iMMX 800	MX816

Zalety rodziny systemów mikrokomputerowych w porównaniu z pojedynczym systemem, z punktu widzenia użytkownika, są szczególnie wyraźne na tle koncepcji przetwarzania rozproszonego, które stanowi dominującą tendencję rozwojową informatyki w ostatnich latach. Mamy tu na myśli budowę i zastosowania lokalnych sieci mikrokomputerowych. Budując taką sieć przy wykorzystaniu Elwro 800 uzyskuje się sieć homogeniczną

dokończenie na s. 19

MICHAŁ KABSCH
Instytut Informatyki, Automatyki i Robotyki
Politechnika Poznańska

Efektywność systemu dBase III

W artykule przedstawiono wyniki badań efektywności najpopularniejszego obecnie mikrokomputerowego systemu zarządzania bazą danych – dBase III. Jak wiadomo system ten jest narzędziem służącym do budowy systemów aplikacyjnych. W zamyśle jego twórców, pakiet dBase III był przeznaczony raczej do niewielkich zastosowań. W Polsce natomiast, ze względu na brak średnich i dużych komputerów, jest często stosowany również do celów przerastających jego możliwości. Z tego względu efektywność systemu użytkowego zrealizowanego z użyciem pakietu dBase III może okazać się nieakceptowalna dla użytkownika.

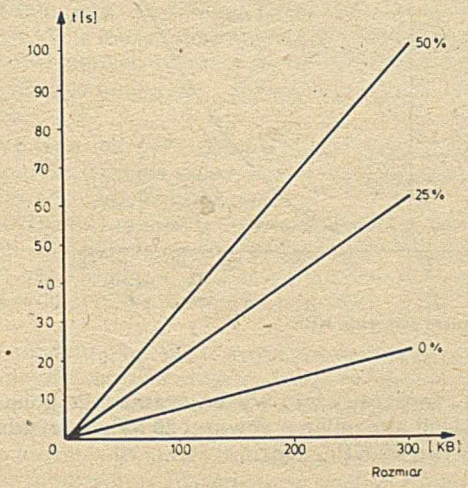
Celem tego artykułu jest przedstawienie wyników badań efektywności systemu dBase III, które pozwolą na oszacowanie efektywności systemu użytkowego już po wstępnym zaprojektowaniu funkcji takiego systemu, określeniu stosowanych w nim poleceń i rozmiarów plików.

Przedstawione wyniki dotyczą czterech podstawowych poleceń systemu dBase III: **LIST** z opcją **FOR** i **WHILE, SORT, INDEX** oraz **JOIN**. Podstawowym celem badań było wyznaczenie czasu wykonywania polecenia w funkcji rozmiaru relacji. Przebadano również zależność czasu wykonywania polecenia od długości i typu atrybutu oraz selektywności, czyli procentowej zawartości w relacji krotek spełniających warunek selekcji.

Pomiary wykonano na mikrokomputerze IBM PC/XT z mikroprocesorem Intel 8088. Operacje dyskowe były wykonane na dysku Winchester o średnim czasie dostępu 80 ms. Rozmieszczenie relacji na dysku było przypadkowe.

EFEKTYWNOŚĆ WYKONYWANIA POLECENIA LIST

Badania efektywności polecenia **LIST** przeprowadzono dla zestawu nieposortowanych relacji o tym samym schemacie, zawierających różną liczbę krotek o długości 250 znaków każda.

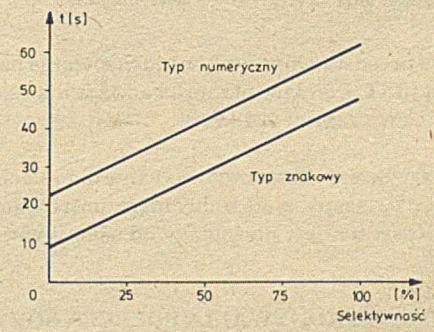


Rys. 1. Efektywność polecenia LIST

Łączna zajętość pamięci przez relacje wahała się od 50 KB do 300 KB. Selektywność wyszukiwania zmieniano w zakresie od 0% do 50%. Wyniki badań przedstawiono na rysunku 1.

Jak widać z wykresu, zależność czasu wykonywania tego polecenia od rozmiaru relacji jest liniowa. Z wykresu wynika również silna zależność czasu wykonywania polecenia od selektywności, co jest spowodowane przetwarzaniem (wyświetlaniem) krotek spełniających warunek selekcji.

W celu pełniejszego zobrazowania efektywności polecenia **LIST** zbadano jej zależność od typu i długości atrybutu oraz wpływ sortowania i indeksowania.



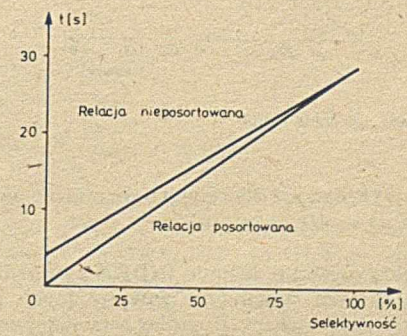
Rys. 2. Wpływ typu danych na efektywność polecenia LIST

Na rysunku 2 przedstawiono czas wykonywania polecenia **LIST** w funkcji selektywności. Polecenie miało postać:

LIST FOR atrybut < stała

W jednym wypadku atrybut był typu numerycznego, a w drugim znakowego. Badanie przeprowadzono dla relacji o rozmiarze 50 KB, zawierającej 984 krotki.

Jak wynika z wykresu, niezależnie od selektywności, wyszukiwanie dla atrybutu numerycznego trwa dłużej o około 10 sek. Jest to wynikiem przeprowadzanej konwersji liczb. Nie stwierdzono natomiast wpływu długości atrybutu (przy tej samej długości krotki) na czas wykonywania polecenia **LIST**.



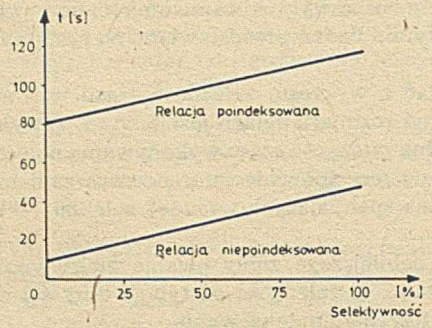
Rys. 3. Wpływ sortowania na efektywność polecenia LIST

Na rysunku 3 przedstawiono wpływ wcześniejszego posortowania relacji na efektywność polecenia LIST. Dla przetwarzania relacji posortowanej polecenie LIST miało postać:

LIST WHILE atrybut < stała

Pomiary przeprowadzono dla relacji o rozmiarze 50 KB, zawierającej 203 krotki.

Jak wynika z wykresu wpływ posortowania na efektywność polecenia **LIST** jest stosunkowo niewielki i maleje w miarę wzrostu selektywności.



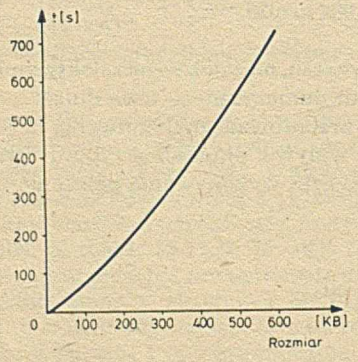
Rys. 4. Wpływ indeksowania na efektywność polecenia LIST

Na rysunku 4 przedstawiono wpływ indeksowania na efektywność polecenia **LIST**. Pomiary przeprowadzono dla relacji o rozmiarze 50 KB, zawierającej 984 krotki.

Jak wynika z wykresu, koszt indeksowania dla relacji o przyjętym rozmiarze, związany z adresowaniem pośrednim, wynosi około 70 sek. Koszt ten jest niezależny od selektywności.

EFEKTYWNOŚĆ WYKONYWANIA POLECEŃ SORT, INDEX I JOIN

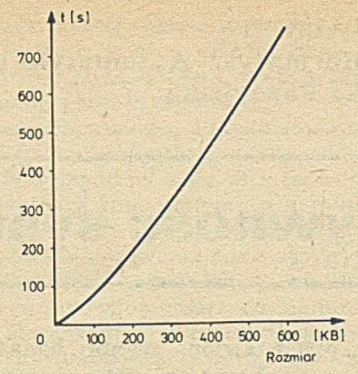
Badanie efektywności polecenia **SORT** przeprowadzono dla zestawu relacji o tym samym schemacie, zawierających różną liczbę krotek o długości 250 znaków. Rozmiar relacji zmieniono od 0 do 600 KB. Wyniki pomiarów przedstawiono na rysunku 5.



Rys. 5. Efektywność polecenia SORT

Jak widać z wykresu już dla relacji o rozmiarze powyżej 50 KB czasy sortowania są rzędu minut.

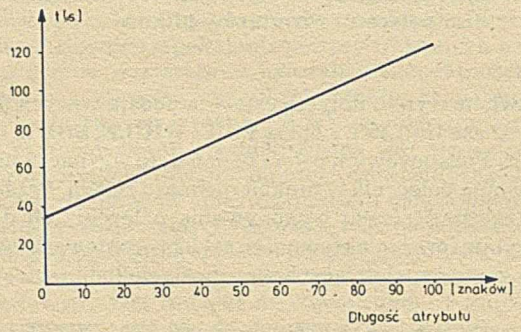
Badanie efektywności polecenia **INDEX** przeprowadzono dla relacji o tym samym schemacie, zawierających różną liczbę krotek o długości 250 znaków. Rozmiar relacji zmieniano od 0 do 600 KB. Długość atrybutu kluczowego wynosiła 50 znaków. Wyniki pomiarów przedstawiono na rysunku 6.



Rys. 6. Efektywność polecenia INDEX

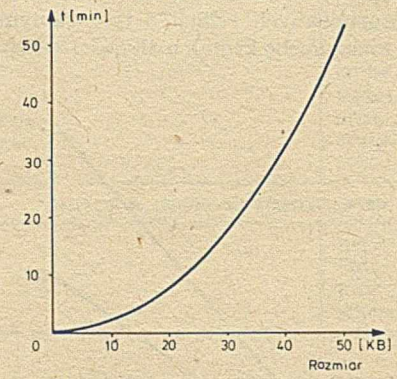
Jak wynika z wykresu charakterystyka efektywności polecenia **INDEX** jest bardzo zbliżona do charakterystyki efektywności polecenia **SORT**. W wypadku polecenia **INDEX** występują jednak dość silne zależności czasu jego wykonywania od długości atrybutu kluczowego. Jest to związane z rozmiarem tworzonego pliku indeksowego.

Zależność czasu wykonywania polecenia **INDEX** od długości atrybutu kluczowego, dla relacji o rozmiarze 100 KB, przedstawiono na rysunku 7.



Rys. 7. Wpływ długości atrybutu kluczowego na efektywność polecenia INDEX

Badanie efektywności polecenia **JOIN** przeprowadzono dla relacji o rozmiarach od 0 do 50 KB. Atrybut połączeniowy był typu znakowego. Wartości tego atrybutu dobrano tak, aby relacje łączone i relacja wynikowa miały ten sam rozmiar. Wyniki pomiarów przedstawiono na rysunku 8.



Rys. 8. Efektywność polecenia JOIN

Jak widać z wykresu czasy wykonywania tego polecenia dla łączonych relacji o rozmiarze powyżej 20 KB sięgają dziesiątek minut, a powyżej 50 KB – godzin.

dokończenie na s. 7

JAN BIELECKI
Warszawa

Turbo C

Modele pamięci, adresowanie, wskazania adresowe

W mikrokomputerach IBM PC adresy pamięci operacyjnej są tworzone z dwóch 16-bitowych części: numeru segmentu i odstępu[1] (ang. *offset*) w segmencie. Jeśli przyjąć, że numer segmentu oznaczono literą **s**, a odstęp oznaczono literą **p**, to adres **a** jest wyznaczany na podstawie formuły:

a = (s << 4) + p

Dokonując tłumaczenia programu na język maszynowy byłoby najwygodniej posłużyć się adresami składającymi się z dwóch danych 16-bitowych: numeru segmentu i odstępu. Nie jest to jednak rozwiązanie optymalne i z tego względu postępuje się zazwyczaj w taki sposób, że część segmentową adresu przechowuje się w rejestrze mikroprocesora, a adresy danych i instrukcji określa się względem adresu wyznaczonego przez część segmentową. Dzięki temu adresy danych i instrukcji programu wynikowego są zaledwie 16-bitowe.

Ograniczenie do posługiwania się odstępami jest w pełni zadowalające w wypadku programów, których ani kod, ani dane, nie przekraczają 64K bajtów pamięci. W wypadku programów dłuższych konieczne jest posługiwanie się pełnymi adresami 32-bitowymi, dotyczącymi tylko danych, tylko kodu, albo zarówno kodu, jak i danych.

W zależności od tego, jaki jest rozmiar kodu oraz danych programu, można dokonać wyboru jednego z sześciu modeli pamięci.

- Model **Tiny** – dane i kod nie mogą zajmować łącznie więcej niż 64 K bajtów pamięci. Adresowanie danych i kodu odbywa się za pomocą wskazań 16-bitowych. Tylko programy oparte na modelu Tiny mogą zostać przetworzone z postaci **.EXE** na postać **.COM**.
- Model **Small** – ani dane, ani kod, nie mogą zajmować więcej niż 64 K bajtów pamięci. Adresowanie danych i kodu odbywa się za pomocą wskazań 16-bitowych.
- Model **Medium** – kod może zajmować więcej niż 64 K bajtów pamięci. Adresowanie kodu odbywa się za pomocą wskazań 32-bitowych, a adresowanie danych odbywa się za pomocą wskazań 16-bitowych.
- Model **Compact** – dane mogą zajmować więcej niż 64 K bajtów pamięci. Kod nie może zajmować więcej niż 64 K bajtów pamięci. Adresowanie danych odbywa się za pomocą wskazań 32-bitowych, a adresowanie kodu odbywa się za pomocą wskazań 16-bitowych.
- Model **Large** – zarówno dane, jak i kod mogą zajmować więcej niż 64 K bajtów pamięci. Adresowanie danych i kodu odbywa się za pomocą wskazań 32-bitowych.
- Model **Huge** – stanowi odmianę modelu Large. Różni się od niego tym, że podczas opracowywania porównań i relacji wskazania są normalizowane. Polega to na takim dobraniu odstępu, aby miał on wartość należącą do przedziału 0..15.

Wybór modelu pamięci, a więc sposobu adresowania danych i kodu, odbywa się poza programem. Przez domniemanie przyjmuje się model Small. Obranie konkretnego modelu powoduje jednolite traktowanie wszystkich wskazań zmiennych i funkcji. Jeśli z pewnych względów jest pożądane potraktowanie wybranego wskazania w inny sposób (np. w modelu Huge, jak wskazania 16-bitowego), to można posłużyć się modyfikatorami adresowania wyrażonymi za pomocą słów kluczowych **near** (bliski), **far** (daleki) i **huge** (odległy).

Każdy z modyfikatorów **near**, **far** i **huge** może występować jedynie w deklaratorze albo w pseudodeklaratorze (tj. deklaratorze, z którego usunięto identyfikator). Jeśli występuje w deklaratorze, to może jedynie poprzedzać identyfikator funkcji albo operator * (gwiazdka). W pierwszym wypadku jawne albo domniemane wystąpienie modyfikatora określa, czy miejsce dla funkcji zostanie przydzielone w domniemanym segmencie danych (near), czy poza nim (far i huge). W drugim wypadku użycie modyfikatora określa, czy związane z nim wskazanie będzie realizowane jako bliskie, dalekie czy odległe. W szczególności, opracowanie deklaracji

```
char huge *Arr [3];
```

spowoduje utworzenie 3-elementowej tablicy, odległych wskazań typu (char).
Jeśli występujący w deklaratorze identyfikator funkcji nie jest poprzedzony modyfikatorem adresowania, to stosuje się domniemania:
near – dla modeli Tiny, Small, Compact,
far – dla modeli Medium, Large, Huge.
Analogicznie postępuje się z operatorami * (gwiazdka) występującymi w deklaratorach i pseudodeklaratorach. W ich wypadku stosuje się domniemania:
near – dla modeli Tiny, Small, Medium,
far – dla modelu Compact, Large, Huge.

Przykład

```
#include <stdio.h>

char huge * Ptr;
char Chr;

union{
  char huge * Ref;
  struct{
    unsigned Off,Seg;
  }Rep;
}Str;

main()
{
  Str.Ref = Ptr = &Chr;
  Str.Rep.Off + = 16;
  Str.Rep.Seg - = 1;
  printf ("%d", Str.Ref = = Ptr);
}
```

Wykonanie programu w dowolnym modelu powoduje wyprowadzenie liczby 1. Wynika stąd, że podczas opracowywania porównań i relacji wskazania dalekie podlegają normalizowaniu. Gdyby program wykonano w modelu Huge, ale z deklaracji zmiennej **Ptr** usunięto modyfikator, to domniemano by modyfikator **far** i wykonanie progamu spowodowałoby wyprowadzenie liczby 0.

Modyfikator near

Dane wskazujące kategorii **near** są reprezentowane w 16 bitach pamięci. W obszarze tym znajduje się jedynie odstęp w segmencie. Numer segmentu znajduje się w rejestrze **DS** albo **CS** procesora, stosownie do tego, czy rozpatrywana dana wskazuje zmienną czy funkcję.

Ponieważ w wypadku danych wskazujących kategorii near numer segmentu jest zawsze taki sam, porównania, relacje i operacje arytmetyczne dotyczące tych wskazań ograniczają się do odstępów. Mimo to, rezultaty porównań i relacji są zawsze zgodne z oczekiwaniami.

Przykład

```
# include <stdio.h>
main
{
  int Fix = 13
    *Ptr = &Fix,
    near *Ref = Ptr;
  printf ("%d", *Ref);
}
```

Jeśli program zostanie skompilowany w modelu Small, to przez domniemanie przyjmie się, że zmienna **Ptr** jest typu **(int near *)** i wykonanie programu spowoduje wyprowadzenie liczby 13. Jeśli program zostanie skompilowany w modelu Large, to przez domniemanie przyjmie się, że zmienna **Ptr** jest typu **(int far *)** i program będzie błędny, a to dlatego, że **Ptr** będzie reprezentować wskazanie dalekie, a zmiennej **Ref** mogą być przypisywane jedynie wskazania bliskie.

Modyfikator far

Dane wskazujące kategorii **far** są reprezentowane w 32 bitach pamięci. W obszarze tym znajduje się zarówno numer segmentu, jak i odstęp w segmencie. Odstęp zajmuje dwa bajty o adresach mniejszych, a numer segmentu zajmuje dwa bajty o adresach większych. Mimo iż dane wskazujące kategorii **far** są reprezentowane w 32 bitach pamięci, jedynie porównania dotyczą danych 32-bitowych, ale traktowanych tak, jakby każda z nich była daną typu **(unsigned long int)** o identycznej reprezentacji. Z tego powodu, a również dlatego, że relacje i operacje arytmetyczne ograniczają się do odstępów, rezultaty porównań i relacji danych wskazujących tej kategorii nie zawsze są zgodne z oczekiwaniami.

Przykład

```
char
Next (char Chr)
{
  return Chr + 1;
}

# include <stdio.h>
main()
{
  char (near *Ref) (char) = Next;
  printf ("%c", (*Ref)('A'));
}
```

Jeśli program zostanie skompilowany w modelu Small, to przez domniemanie przyjmie się, że funkcja **Next** jest typu:

```
(char near (char))
```

i wykonanie programu spowoduje wyprowadzenie litery **B.** Jeśli program zostanie skompilowany w modelu Large, to przez domniemanie przyjmie się, że funkcja **Next** jest typu

```
(char far (char))
```

i program będzie błędny, a to dlatego, że Next będzie reprezentować wskazanie dalekie, a zmiennej **Ref** mogą być przypisywane jedynie wskazania bliskie.

Modyfikator huge

Dane wskazujące kategorii **huge** są reprezentowane w 32 bitach pamięci. W obszarze tym znajduje się zarówno numer segmentu, jak i odstęp w segmencie. W odróżnieniu do wskazań kategorii **far**, wskazania kategorii **huge** są normalizowane, a porównania, relacje i operacje arytmetyczne dotyczące tych wskazań nie ograniczają się do odstępów i dotyczą także numerów segmentów. Dzięki temu rezultaty porównań i realizacji są zawsze zgodne z oczekiwaniami, a za pośrednictwem wskazań rozpatrywanej kategorii można odwoływać się do obiektów zajmujących więcej niż 64 K bajtów pamięci operacyjnej.

Przykład

```
# include <stdio.h>
union{
  int far *Var;
  struct{
    unsigned Off,Seg;
  } Str
} Ref,Ptr;
int Val = 13;

main()
{
  Ref.Var = Ptr.Var = &Val;
  Ref.Str.Off - = 32;
  Ref.Str.Seg + = 2;
  printf ("%d,%d,%d", *Ref.Var,
                      Ref.Var) = Ptr.Var,
                      *Ptr.Var);
}
```

Jeśli program zostanie skompilowany dla modelu Small, to jego wykonanie spowoduje wyprowadzenie napisu **13,0,13.** Jeśli zostanie skompilowany dla tego samego modelu, ale modyfikator **far** zostanie zmieniony na **huge,** to nastąpi wyprowadzenie napisu **13,1,13.**
Oto kilka dodatkowych deklaracji:
A. Jeśli obrano model pamięci Small, to **Ptr** jest zmienną wskazującą, której mogą być przypisywane odległe wskazania danych typu **(char),** a **Ref** jest zmienną wskazującą, której mogą być przypisywane bliskie wskazania takich zmiennych, którym mogą być przypisywane dalekie wskazania zmiennych typu **(int).**

```
char huge * Ptr;
int far * *Ref;
```

B. Jeśli obrano model pamięci Large, to **Ptr** jest zmienną wskazującą, której mogą być przepisywane dalekie wskazania zmiennych, którym mogą być przypisywane odległe wskazania zmiennych typu **(char):**

```
char huge * *Ptr;
```

C. Niezależnie od obranego modelu, **Ptr** jest zmienną wskazującą, której mogą być przypisywane bliskie wskazania takich zmiennych, kórym mogą być przypisywane dalekie wskazania zmiennych typu **(char),** a **Ref** jest zmienną, której mogą być przypisywane dalekie wskazania zmiennych typu **(unsigned int).**

```
char far * near * Ptr;
unsigned far * far * Ref;
```

D. Na skutek przyjętych ograniczeń co do dopuszczalnych miejsc wystąpienia modyfikatorów **near. far** i **huge,** w języku Turbo C błędne są następujące deklaracje:

float Arr[20000]; – ponieważ tablica Arr zajmuje więcej niż 32 K bajtów pamięci.
char far Brr[20]; – ponieważ nie implementowano dalekich zmiennych, a jedynie dalekie wskazania.
Należy odnotować, że przytoczone deklaracje błędne są poprawne w sensie implementacji MicroSoft – wersja 5.0.

Jak wynika z podanego opisu, w implementacji Turbo C jedynie funkcje mogą być zadeklarowane jako bliskie, dalekie i odległe. Zadeklarowanie funkcji jako bliskiej przyspiesza odwoływanie się do niej, ale ogranicza je do modułu źródłowego, w którym została zadeklarowana. W odróżnieniu od funkcji zmienne reprezentowane przez identyfikatory są zawsze bliskie.

Przykład

```
#include <stdio.h>
main()
{
  int near Fact(int);
  printf("%d", Fact(5));
}

int near
Fact (int n)
{
  if(n) return Fact(n - 1) * n;
  return 1;
}
```

Jeśli program zostanie skompilowany w modelu Medium, Large albo Huge, to będzie wykonywany szybciej, ponieważ wywołania funkcji **Fact** będą realizowane jako 16-bitowe.

Należy podkreślić, że klasyfikacja wskazań na bliskie, dalekie i odległe dotyczy nie tylko wskazań zmiennych i funkcji, ale w równym stopniu dotyczy wskazań adresowych. W szczególności opracowanie dla modelu Small deklaracji

```
void far * Ptr;
```

spowoduje utworzenie zminnej wskazującej, której będą mogły być przypisywane wskazania adresowe lokalizujące dalekie obszary pamięci operacyjnej.

Przykład

```
  include <stdio.h>
  include <alloc.h>
  define SIZE 20000
void huge * Ptr,huge * Ref;
main()
{
  int Count;
  long Sum = 0;
  if (coreleft() < (long) (SIZE) * sizeof (long)){
  printf ("Unavailable memory");
  return;
}
  Ptr = calloc (SIZE, sizeof (long));
  for(Count = 0; Count < SIZE; Count++){
    Ref = Ptr;
    *(long *)Ptr = Count;
    Ptr = (void *) ((long huge *) Ptr + 1;
    if(Ref > Ptr){
      printf ("Addressing error");
      return;
    }
  }

  for(Count = 0; Count < SIZE; Count++){
    Ptr = (void *) ((long huge *)Ptr - 1);
    Sum += *(long *)Ptr;
  }
  printf ("Sum = %ld", Sum);
}
```

Przebieg wykonania programu zależy od obranego modelu pamięci. Dla modelu Small następuje wyprowadzenie napisu:

Unavailable memory

Dla modelu Large i Huge następuje wyprowadzenie sumy nieujemnych liczb całkowitych z przedziału **0..20000**. Należy zauważyć, że gdyby występujące w programie operatory konwersji:

(long huge *)

zostały zastąpione operatorami bez modyfikatora **huge**, to nawet w modelu Huge wykonywanie programu zostałoby zakończone wyprowadzeniem napisu:

Addressing error

Wynika to stąd, że takie operatory byłyby traktowane jak operatory:

(long far *)

a inkrementowanie wskazań zostałoby ograniczone do przemieszczeń.

System mikrokomputerowy Elwro 800

dokończenie ze s. 14

z punktu widzenia zastosowanych rozwiązań w dziedzinie sprzętu i oprogramowania podstawowego, a heterogeniczną z punktu widzenia funkcjonalnego poszczególnych stacji – specjalizowane stacje dostępu, przekazywania danych, drukowania, graficzne, sterujące itp.

Konstrukcja całej rodziny systemów mikrokomputerowych ma również duże znaczenie dla producenta. Produkuje on bowiem ograniczony zestaw typowych modułów, które następnie wchodzą w skład różnorodnych systemów. W systemie Elwro 800 te same moduły pamięci, mikrokomputerów sterujących oraz sterowników biernych są wykorzystywane do konfigurowania systemów ośmio– i szesnastobitowych, jedno– i wieloprocesorowych. Ponadto jednorodność i masowość produkcji mają duże znaczenie dla ceny, niezawodności, czasu dochodzenia do docelowego poziomu produkcji itp.

* * *

Jak wynika z przedstawionych rozwiązań i analizy specyficznych wymagań rynku polskiego, jednym z głównych celów przyjęcia takiej a nie innej koncepcji systemu mikrokomputerowego Elwro 800, było dążenie do dostarczenia na ten rynek systemu bazowego, który byłby podstawą rozwoju różnorodnych zastosowań, zarówno personalnych, jak i przemysłowych. Elwro 800 jest systemem rekonfigurowalnym i otwartym, co stanowi jego podstawową zaletę. W naturalny sposób można dołączyć do niego specjalizowane moduły wymagane dla konkretnych zastosowań. System Elwro 800 umożliwia zatem skierowanie wysiłku informatyków, konstruktorów systemów mikrokomputerowych na zastosowania, przez odciążenia ich od konieczności konstruowania modułów podstawowych oraz przerabiania systemów zamkniętych i dostosowywania ich niejako na siłę do spełnienia wymagań, do których nie są dostosowane.

Jedna z konfiguracji Elwro 800 – konfiguracja jednoprocesorowa z mikrokomputerem szesnastobitowym – jest zgodna z IBM PC/XT. W konfiguracji tej zainstalowany jest odpowiednik systemu operacyjnego PC-DOS, co umożliwia wykorzystanie całego oprogramowania firmy IBM. Największa zatem zaleta mikrokomputera IBM PC/XT – oprogramowanie – została uwzględniona w systemie Elwro 800. Nie przejęto przy tym wad tego mikrokomputera, głównie zamkniętości struktury, jednoprocesorowości, ograniczenia pojemności pamięci opreracyjnej do 640 KB itd.

ROMAN PODRAZA
Instytut Informatyki
Politechnika Warszawska

Sterowanie przepływowe (2)

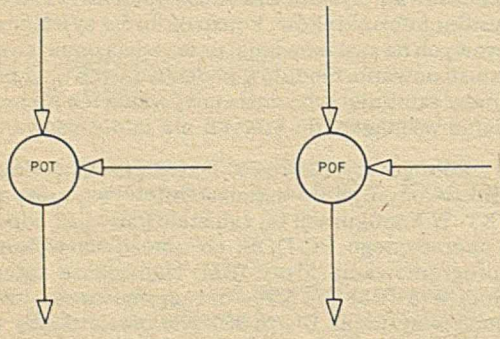
Implementacja modelu

W latach siedemdziesiątych i osiemdziesiątych pojawiły się pierwsze propozycje systemów sterowania przepływowego [1, 11, 12, 13, 14], a w ślad za nimi kilka prototypowych architektur przeznaczonych do praktycznej weryfikacji przydatności tego modelu. Pierwsze maszyny systemów sterowania przepływowego wytworzone w ośrodkach naukowych jak MIT [5] w USA, CERT-ONERA [4] w Tuluzie (Francja), NTT [15] (Japonia) są urządzeniami eksperymentalnymi, przeznaczonymi do sprawdzenia różnych koncepcji naukowych, weryfikacji przyjętych rozwiązań technicznych i zdobycia doświadczeń praktycznych. Urządzenia te podlegają nieustannym modyfikacjom i rozbudowie. Jedną z wiodących konstrukcji systemów sterowania przepływowego jest architektura zaprojektowana i wykonana na Uniwersytecie w Manchester [6, 7, 9 16]. Działa ona od 1981 r. i jest typowym przykładem architektur dynamicznych [10].

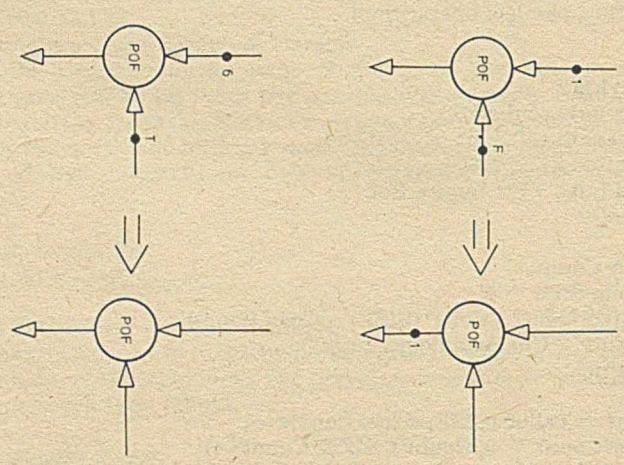
Oprócz ekperymentalnych konstrukcji systemów sterowania przepływowego handlowo dostępne są elementy przetwarzające z pamięcią i układami do komunikacji, które można łączyć i programować korzystając z idei sterowania przepływowego (transputery firmy Inmos programowane w języku Occam). Firma NEC wyprodukowała układ scalony o symbolu μPD7281, będący specjalizowanym procesorem do przetwarzania obrazów. Jest on reklamowany jako urządzenie, które dzięki zastosowaniu sterowania przepływowego przewyższa – pod względem szybkości przetwarzania – jego konwencjonalny odpowiednik wykonany w tej samej technologii. Bliższe dane na temat jego konstrukcji nie są jednak dostępne ze względu na tajemnicę przemysłową.

FUNKCJE MASZYNY Z MANCHESTERU

Maszyna sterowania przepływowego skonstruowana na Uniwersytecie w Manchester ma architekturę wieloprocesorową przeznaczoną do uniwersalnego zastosowania. Model sterowania przepływowego realizowany przez ten system ma pewne ograniczenia w stosunku do opisu przedstawionego w pierwszej części artykułu. Każdy węzeł elmentarny (któremu odpowiada instrukcja maszynowa) może mieć najwyżej dwa łuki wejściowe i dwa łuki wyjściowe. Węzły reprezentujące funkcje o więcej niż dwóch wejściach mogą być definiowane z węzłów elementarnych, a więc ograniczenie liczby łuków wejściowych i wyjściowych nie zmniejsza możliwości funkcjonalnych modelu (i systemu). Ponieważ selektor ma 3 łuki wejściowe, więc nie spełnia ograniczenia narzuconego przez manchesterską wersję modelu sterowania przepływowego. Zamiast selektora i współpracującego z nim dystrybutora (umożliwiających realizację obliczeń warunkowych **if..then..else** oraz pętli) w modelu manchesterskim występują dwuwejściowe węzły **POT** (ang. *pass-on-true*) i **POF** (ang. *pass-on-false*) przedstawione na rys. 1. Poziomy łuk wejściowy jest łukiem sterującym i znacznik na tym łuku powinien mieć wartość logiczną. Odpalenie węzła **POT** polega na tym, że jeśli znacznik na łuku sterującym ma wartość logiczną „prawda", to znacznik z pionowego łuku wejściowego jest kopiowany na łuk wyjściowy. Jeśli natomiast znacznik na łuku sterującym ma wartość „fałsz", to po odpaleniu węzła znikają znaczniki z łuków wejściowych (te, które spowodowały przygotowanie węzła), ale na łuku wyjściowym nie pojawia się żaden znacznik. Reguła odpalania dla węzła **POF** jest analogiczna, przy czym wartość „fałsz" na łuku sterującym powoduje „przepuszczenie" znacznika z pionowego łuku wejściowego przez węzeł, a wartość „prawda" – anihilację tego znacznika (rys. 2).



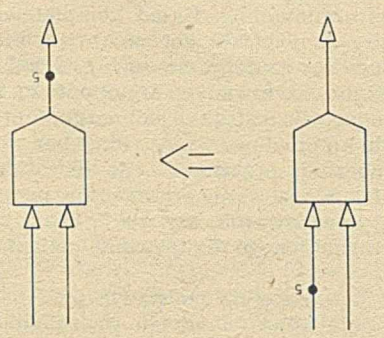
Rys. 1. Węzły sterujące w modelu manchesterskim: POT (*pass-on-true*) i POF (*pass-on-false*)



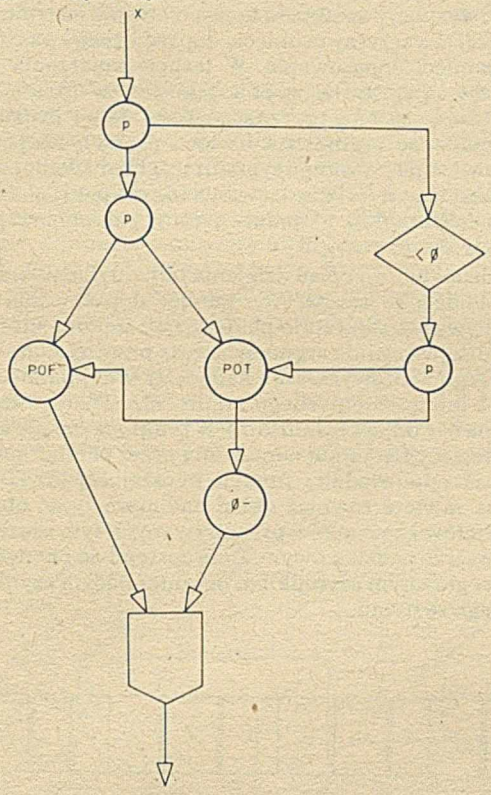
Rys. 2. Odpalenia węzła POF dla wartości logicznej „fałsz" (F) i dla wartości „prawda" (T) występujących na łuku sterującym

Z węzłami **POT** i **POF** współpracuje węzeł **fuzja** (ang. *merge*), przedstawiony na rys. 3. Węzeł ten jest przygotowany, jeśli na jednym z jego dwóch łuków wejściowych pojawi się znacznik. Odpalenie węzła powoduje skopiowanie znacznika z łuku wejściowego na łuk wyjściowy. Jeśli na obu łukach wejściowych fuzji są znaczniki, to są one kopiowane na łuk wyjściowy w dowolnej kolejności. W modelu manchesterskim zakłada się, że nie jest możliwe pojawienie się znaczników (tego samego koloru) jedno-

cześnie na obu łukach fuzji. Założenie to pozwala pominąć strategię wyboru kolejności kopiowania znaczników z łuków wejściowych; jest ono zawsze spełnione dla grafów generowanych przez programy zapisane w języku wysokiego poziomu, dostosowanym do modelu sterowania przepływowego (LAPSE [8] lub SISAL [10]). Na rysunku 4 pokazano sposób organizacji obliczania wyrażenia **if..then..else** do realizacji funkcji wartości bezwzględnej.



Rys. 3. Odpalenie węzła fuzja



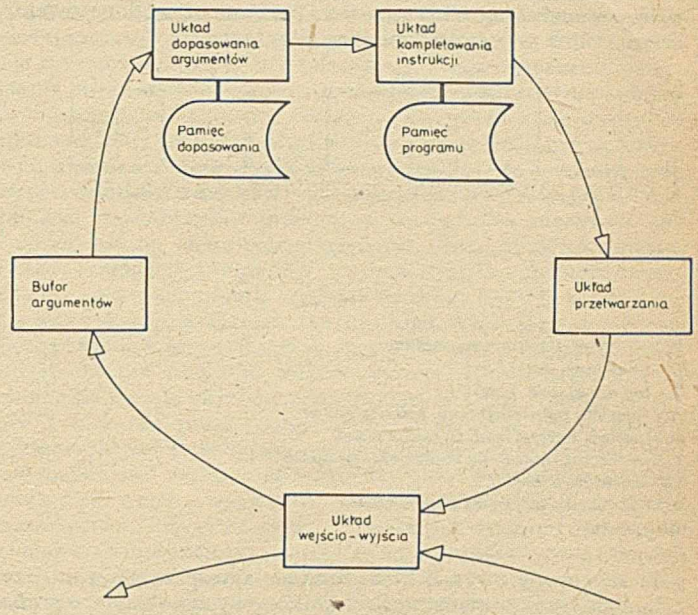
Rys. 4. Zastosowanie węzłów POT, POF i fuzji do realizacji obliczania wartości bezwzględnej według wyrażenia:
if x<0 then x else x

Z fuzją może też współpracować dystrybutor, który w modelu manchesterskim nazywa się **rozgałęzieniem** (ang. *branch*). Odpalenia węzłów sterujących w modelu manchesterskim stanowią pewien wyjątek w ogólnej regule. Odpalenie węzła **POT** (albo **POF**) może nie generować żadnego znacznika na łuku wyjściowym. Węzły elementarne z modelu manchesterskiego odpowiadają poziomowi instrukcji asemblerowych w klasycznym systemie komputerowym. Ograniczenie do dwóch argumentów nie jest więc niczym zadziwiającym na tym poziomie.

ARCHITEKTURA MASZYNY

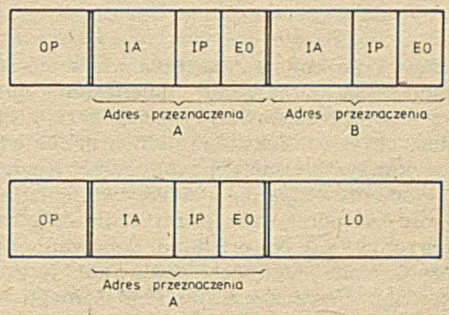
Architektura manchesterska reprezentuje klasę maszyn potokowych (ang. *pipeline*). Maszyna jest złożona z pięciu autonomicznych układów (rys. 5):

- wejścia-wyjścia,
- buforowania argumentów,
- dopasowania argumentów,
- kompletowania instrukcji,
- przetwarzania.



Rys. 5. Schemat architektury systemu sterowania przepływowego (na rysunku zamiast napisu „bufor argumentów" powinien wystąpić napis „układ buforowania argumentów")

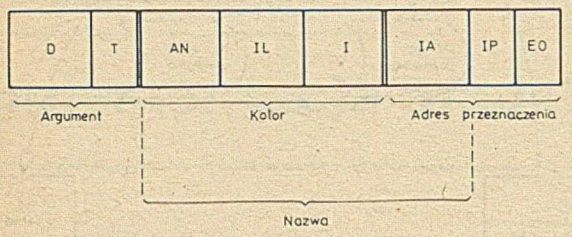
Każdy układ pracuje niezależnie, według własnego lokalnego zegara. Informacje między układami są wymieniane przez asynchroniczne transmisje. Każdy z układów ma podwójne bufory do wysyłania i odbierania informacji. Parametry techniczne układów są ze sobą zgodne, a więc czasy przetwarzania są zbliżone. Protokół transmisji informacji między układami gwarantuje, że nie dojdzie do utraty informacji w sytuacji, gdy jeden z układów generuje informacje szybciej niż może je przyjąć odbiorca. Niemożność wysłania informacji powoduje zawieszenie pracy danego układu, aż do zwolnienia buforów wyjściowych. Tak więc architektura ta nie jest dostosowana do tolerowania uszkodzeń sprzętowych. Dopiero jej rozbudowa pozwala uzyskać tę pożądaną cechę.



Rys. 6. Formaty instrukcji przechowywanej w pamięci programu:
OP – operacja (ang. *operation*)
IA – adres instrukcji (ang. *instruction address*)
IP – punkt wejścia (ang. *input point*); prawy lub lewy argument
EO – liczba oczekiwanych argumentów (ang. *expected operands*)
LO – literał (ang. *literal operand*)

Graf sterowania przepływowego jest reprezentowany przez instrukcje umieszczone w **pamięci programu** (grafu), będącej częścią **układu kompletowania instrukcji.** Format instrukcji przedstawiono na rys. 6. Instrukcja reprezentuje węzeł elementarny i zawiera informacje o operacji oraz o łukach wyjściowych węzła (adresy przeznaczenia **A** i **B**). W instrukcji może być zawarta wartość literału, jeśli instrukcja reprezentuje węzeł o jednym łuku wyjściowym (literał zastępuje adres przeznaczenia **B**). Łuki wyjściowe węzła są zdefiniowane przez wskazanie,

do których węzłów (pole **IA**) oraz do którego z dwóch możliwych punktów wejść prowadzą (pole **IP** określające lewy lub prawy argument). Dla węzłów z jednym łukiem wyjściowym jest używany tylko adres przeznaczenia **A**; adres przeznaczenia **B** może być nie wykorzystany. Pole **EO** określa liczbę łuków wejściowych węzła, do którego prowadzi łuk zdefiniowany przez pole przeznaczenia. Z informacji tej korzysta układ dopasowania argumentów przy kojarzeniu argumentów.



Rys. 7. Format pakietu znacznika:
D – dana (ang. *data*)
T – typ danej (ang. *type*)
AN – poziom rekurencji (ang. *activity name*)
IL – poziom iteracji (ang. *iteration lewel*)
I – indeks(ang. *index*); do implementacji tablic
IA – adres instrukcji
EO – liczba oczekiwanych argumentów
LO – literał

W schemacie z rys. 5 bloki funkcjonalne przesyłają między sobą pakiety informacyjne. Znacznikom występującym w grafie sterowania przepływowego odpowiadają **pakiety znaczników** (rys. 7). Pakiet znacznika zawiera informacje o wartości i typie argumentu, o kontekście obliczeń dla danego znacznika i o adresie przeznaczenia znacznika. Kontekst obliczeń (i odpowiadający mu kolor znacznika) uwzględnia trzy różne źródła równoległości obliczeń. Dlatego pole koloru jest podzielone na 3 części. Pierwsza część pola koloru służy do rozróżnienia znaczników z różnych poziomów rekurencji (pole **AN**). Druga część (pole **IL**) – służy do rozróżniania znaczników z różnych kroków iteracji.

Przy zastrzeżeniu, że w pętli nie może wystąpić kolejna pętla oraz, że rekurencyjne wywołanie dotyczy nie więcej niż jednej funkcji naraz, odpowiednie pola mogłyby być modyfikowane przez prostą inkrementację (dekremetację) w celu jednoznacznego definiowania kontekstu znacznika. Ponieważ zastrzeżenie to jest nie do przyjęcia, ze względu na ograniczenia, jakie narzuca na sposób programowania, do modyfikacji tych pól muszą być stosowane inne techniki. W maszynie manchesterskiej wykorzystuje się specjalny procesor w układzie przetwarzania [7].

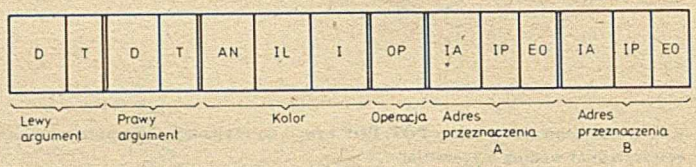
Trzecia część pola koloru znacznika wiąże się ze sposobem implementacji tablic w maszynie manchesterskiej. Każdy element tablicy jest reprezentowany przez znacznik, którego trzecie pole koloru określa indeks tego elementu. Takie rozwiązanie umożliwia wykonywanie operacji tablicowych równocześnie dla wszystkich elementów zgodnie z indeksowaniem (oczywiście ta równoczesność jest teoretyczna, praktycznie ograniczona możliwościami sprzętowymi). Na przykład, dodawanie dwóch tablic powoduje, że w grafie jeden węzeł dodawania jest przygotowywany i odpalany przez znaczniki tego samego koloru, tzn. znaczniki z wartościami elementów tablic z tymi samymi indeksami. Adres przeznaczenia znacznika określa łuk, na którym się on znajduje. Łuk ten jest zdefiniowany przez wskazanie instrukcji w pamięci programu (pole **IA**) oraz punktu wejścia (pole **IP**). Pakiet znacznika zawiera też informację o liczbie argumentów przewidzianych dla operacji, do której jest przeznaczony. Pakiety znaczników są generowane przez układ przetwarzania lub są wprowadzane z zewnętrz (dane wejściowe) przez układ wejścia-wyjścia. Układ wejścia-wyjścia służy również do wysyłania pakietów znaczników z rezultatami na zewnętrz.

Pakiety znaczników gromadzą się w **układzie buforowania argumentów**, którego zadaniem jest przechowywanie pakietów znaczników w systemie. Nie można bowiem określić szybkości (w krótkich odcinkach czasu) generowania pakietów znaczników przez wieloprocesorowy układ przetwarzania.

Z buforu argumentów pakiety znaczników są przesyłane do **układu dopasowania argumentów.** Jest to układ o kluczowym znaczeniu dla całego systemu. Jego zadaniem jest wyszukanie pakietów znaczników przenoszących argumenty dla danej instrukcji i utworzenie z nich **pakietu kompletu argumentów.** Dla instrukcji jednoargumetowej (węzła jednowejściowego) pakiet kompletu argumentów jest po prostu pakietem znacznika. Natomiast dla instrukcji dwuargumentowej, pakiet kompletu argumentów ma format zbliżony do pakietu znacznika, ale zawiera dwie wartości argumentów. Układ kompletowania instrukcji rozpoznaje, czy dany pakiet znacznika jest przeznaczony do instrukcji jedno – czy dwuargumentowej, na podstawie zawartości pola **EO** (liczba oczekiwanych argumentów). Pakiet znacznika przeznaczonego dla instrukcji jednoargumentowej jest wysyłany do układu kompletowania instrukcji bez zmian. W wypadku pakietu znacznika przeznaczonego dla instrukcji dwuargumentowej układ dopasowania argumentów przeszukuje asocjacyjną **pamięć dopasowania**, czy nie jest tam zapisany pakiet znacznika przeznaczonego dla tej samej instrukcji.

Pakiety są kojarzone na podstawie pola nazwy, na które składają się pola koloru i adresu instrukcji (rys. 7). Jeżeli w pamięci asocjacyjnej znajduje się pakiet znacznika z nazwą obsługiwanego pakietu, to jest tworzony pakiet kompletu argumentów i wysyłany do układu kompletowania instrukcji. Opisanej operacji towarzyszy usunięcie dopasowanego pakietu znacznika z pamięci dopasowania. W trakcie kojarzenia pakietów znaczników są sprawdzane pola punktów wejścia (**IP**) w obu pakietach, czy są one przeznaczone do danej instrukcji jako komplementarne argumenty (lewy i prawy). Niepowodzenie tego sprawdzenia powoduje sygnalizację błędu wykonania programu. Jeżeli w pamięci dopasowania nie ma pakietu komplementarnego argumentu, to przetwarzany pakiet znacznika jest wpisywany do tej pamięci.

Z przedstawionego opisu działania układu dopasowania argumentów widać, że asocjacyjna pamięć dopasowania powinna przechowywać te wszystkie pakiety znaczników, które dotarły do układu dopasowania argumentów, są przeznaczone do instrukcji dwuargumentowych i nie napotkały w tej pamięci pakietu znacznika komplementarnego argumentu. Dlatego też pamięć ta powinna być odpowiednio duża. W praktyce stosuje się pamięć pseudoasocjacyjną, implementowaną przez pamięć adresowaną za pomocą odpowiedniej funkcji mieszającej (ang. *hashing*). Pamięć ta zawiera również układ nadmiarowy do obsługi różnych pakietów znaczników przypisywanych tym samym pozycjom w obszarze podstawowym. Czas dostępu do pamięci dopasowania jest głównym czynnikiem ograniczającym szybkość działania całego systemu.



Rys. 8. Format instrukcji wykonawczej: OP – operacja, pozostałe oznaczenia jak dla formatu pakietu znacznika (rys. 7)

Układ kompletowania instrukcji uzupełnia pakiet kompletu argumentów o kod operacji, która ma być wykonana, oraz o adresy przeznaczenia dla pakietów znaczników rezultatów wykonania operacji. Informacje te są pobierane z pamięci programu, gdzie jest przechowywany opis grafu. Pakiet kompletu argumentów zawiera m. in. adres instrukcji, dla której jest przeznaczony. Informacja ta jest wykorzystywana w odwołaniu do pamięci programu. W wyniku działania układu kompletowania instrukcji powstaje **pakiet instrukcji wykonawczej** (rys. 8). Dla instrukcji reprezentującej węzeł mający mniej niż dwa łuki wejściowe lub mniej niż dwa łuki wyjściowe, niektóre z pól pakietu instrukcji wykonawczej mogą być niewykorzystane (np. prawy argument lub adres przeznaczenia **B**).

Układ przetwarzania składa się z kilku równolegle połączonych mikroprogramowalnych bloków przetwarzania. Każdy

z nich może wykonać dowolną operację, na podstawie dostarczonego pakietu instrukcji wykonawczej. W wyniku mogą powstać jeden lub dwa pakiety znaczników (zależy to od liczby łuków wyjściowych odpowiedniego węzła w grafie) albo może nie powstać żaden z pakietów (np. odpalenie węzła POT z wartościami „fałsz" na łuku sterującym). Liczba procesorów w układzie przetwarzania jest tak dobrana, aby średni czas przetwarzania pakietu instrukcji wykonawczej w układzie był zgodny z parametrami czasowymi pozostałych układów systemu.

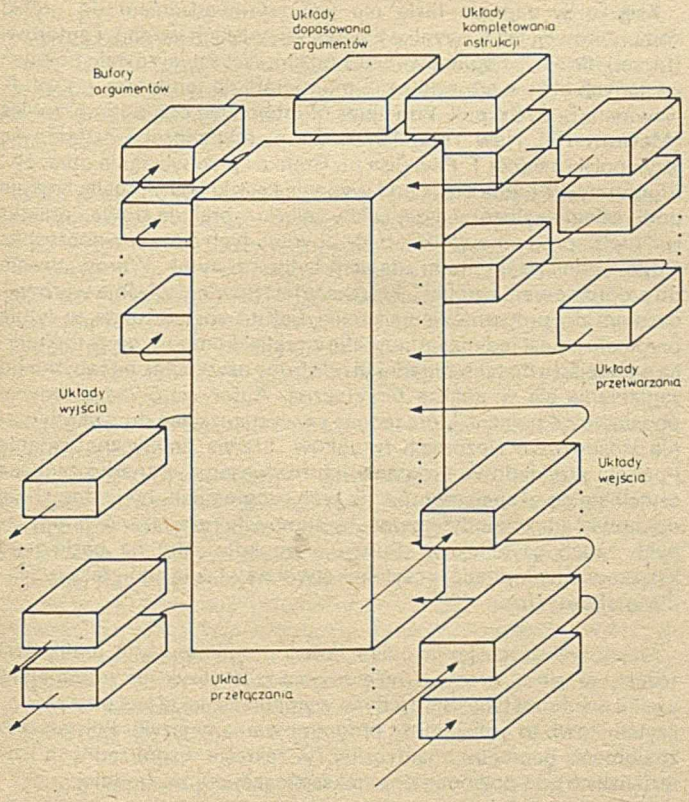
Pakiety znaczników z układu przetwarzania są przesyłane do **układu wejścia-wyjścia.** Układ ten organizuje wymianę pakietów znaczników z otoczeniem.

ZASADA DZIAŁANIA

Maszyna manchesterska pracuje w dwóch trybach: **inicjowania** i **wykonania.** W trakcie inicjowania są wykonywane trzy zadania. Najpierw jest opróżniany układ buforowania argumentów i pamięć dopasowania. Następnie instrukcje programu są ładowane do pamięci programu. W końcu zbiór pakietów znaczników z danymi wejściowymi (odpowiadający znacznikom na łukach wejściowych grafu) jest przekazywany z zewnętrz do układu wejścia-wyjścia.

Na początku trybu wykonania pakiety znaczników z danymi wejściowymi są wprowadzone do systemu przez układ wejścia-wyjścia i gromadzone w układzie buforowania argumentów. Działanie systemu jest następnie zdeterminowane przez stany pamięci dopasowania i pamięci programu. Zakończenie działania programu następuje wówczas, gdy układy dopasowania argumentów, kompletowania instrukcji i przetwarzania są bezczynne, a bufor argumentów jest pusty. W tym stanie wszystkie pakiety znaczników z rezultatami działania programu powinny być już przekazane do otoczenia przez układ wejścia-wyjścia.

Otoczenie maszyny manchesterskiej stanowi konwencjonalny komputer spełniający rolę nadzorcy. Jego zadaniem jest zapewnienie inicjowania maszyny manchesterskiej oraz dostarczenie danych i odebranie wyników w trakcie działania maszyny w trybie wykonania.



Rys. 9. Schemat rozbudowanej architektury manchesterskiego systemu sterowania przepływowego

Niezawodność proponowanej architektury nie jest jej mocną stroną, ponieważ mogą być tolerowane jedynie uszkodzenia procesorów w układzie przetwarzania. Każda niesprawność innego układu uniemożliwia pracę całej maszyny. Planuje się rozbudowanie architektury do wielu pierścieni połączonych jednym rozbudowanym układem wejścia-wyjścia, zwanym **układem przełączania** (ang. *exchange switch*). Schemat takiej rozbudowanej architektury przedstawiono na rys. 9. Uszkodzenia sprzętowe mogą tu być tolerowane przez wyłączanie pierścienia, w którym występiło uszkodzenie. Wraz z rozbudową architektury wzrasta moc obliczeniowa systemu. Krytycznym układem staje się układ przełączania, odpowiedzialny za wprowadzanie danych i wyprowadzanie rezultatów do otoczenia oraz za przesyłanie znaczników między pierścieniami systemu. Układ ten jest najsłabszym ogniwem ze względu na niezawodność i odporność na błędy całego systemu. Wraz ze zwiększeniem liczby pierścieni i układów wejścia-wyjścia, obsługiwanych przez układ przełączania, wydłuża się droga pakietu znacznika, a więc i czas jego przesyłania przez układ. Wiąże się to z większą liczbą układów decyzyjnych służących do wyboru odpowiednich pierścieni lub układów wejścia-wyjścia. Układ przełączania składa się z wielu układów przełączających i uszkodzenie któregoś z nich nie musi powodować wstrzymania pracy całego systemu. Konsekwencją takiego uszkodzenia może być przerwanie możliwości przesyłania pomiędzy niektórymi pierścieniami i układami wejścia-wyjścia. Uszkodzenie takie może być tolerowane przez odłączenie części składowych systemu lub przez znalezienie innej ścieżki połączeń z wykorzystaniem pośredniego pierścienia. Uszkodzeniem, przed którym system nie ma zabezpieczenia, jest przekłamywanie układu przełączania powodujące niewłaściwe przesyłanie pakietów znaczników. Rozwiązanie tego problemu przez zastosowanie sprzętowych technik wykrywania i tolerowania uszkodzeń przy projektowaniu układu przełączania, powinno w efekcie uczynić z architektury manchesterskiej system o wysokim stopniu odporności na uszkodzenia. Kosztem tolerancji uszkodzeń jest spadek wydajności i maksymalnej szybkości przetwarzania systemu.

Wydajność systemów wieloprocesorowych jest wielkością bardzo istotną w ocenie nakładów przeznaczonych na rozbudowę systemu i uzyskiwanych efektów. W wypadku manchesterskiego pierścienia (ogólnie architektury typu potokowego) wydajność systemu ściśle wiąże się z jego zajętością. Jeżeli podczas pracy pierścień jest zajęty w 100% (tzn. żaden element nie pracuje „na próżno"), to szybkość przetwarzania zależy od częstotliwości zegara. W praktyce niektóre bloki pierścienia nie wykonują użytecznych czynności czekając na przesłanie informacji. Okazuje się, że dla programu, którego równoległość jest większa niż równoległość pierścienia, gwarantowane jest, że zajętość pierścienia będzie większa niż 1/2 (50%). Przy pełnej zajętości pierścienia zwiększenie szybkości przetwarzania jest proporcjonalne do liczby procesorów w układzie przetwarzania (procesory te nie muszą odwoływać się do pamięci po kod i argumenty, dzięki czemu każdy z nich wykonuje operacje szybciej niż w konwencjonalnym komputerze). Pozostałe czynności w systemie wykonywane równolegle służą do organizacji przebiegu obliczenia.

Jeżeli w systemie jest **k** procesorów działających z szybkością **M** rozkazów na sekundę, to dla pełnej zajętości pierścienia system może przetwarzać **k*M** rozkazów na sekundę. W rzeczywistości trzeba ten wynik pomnożyć przez wskaźnik zajętości pierścienia (mniejszy niż 1), który uwzględnia opóźnienia dostarczania informacji do układu przetwarzania. Dla rozbudowanej architektury manchesterskiej gwarantowany wskaźnik zajętości systemu dla programu o stopniu równoległości nie mniejszym niż równoległość sprzętu wynosi co najmniej **1/n**, gdzie **n** jest liczbą pierścieni. Na przykład, dla 4 pierścieni wskaźnik ten wynosi 25%, przy czym równoległość sprzętu w tym wypadku jest bardzo duża i taka równoległość w programie może być uzyskana dzięki rekurencji, iteracji i operacjom tablicowym. Niemożność uzyskania większych współczynników zajętości architektury manchesterskiej wynika ze sposobu pracy układu dopasowania argumentów i układu przetwarzania. Układ dopasowania argumentów co jakiś czas nie produkuje informacji wyjściowej w odpowiedzi na informacje wejściową (wpisuje

wtedy pakiet znacznika do pamięci dopasowania). Procesor w układzie przetwarzania może wygenerować dwa, jeden albo żadnego pakietu znacznika. Przytoczone szacunkowe miary wydajności są niewielkie w porównaniu z miarami wydajności dla konwencjonalnych systemów jednoprocesorowych, ale są one charakterystyczne dla różnych systemów wieloprocesorowych.

* * *

W przedstawionym opisie implementacji modelu sterowania przepływowego pominięto wiele rozwiązań służących m. in. do organizacji programowania niedeterministycznego [2], [3] i do obsługi tablic [8]. Układ dopasowania argumentów może kompletować pakiety znaczników na 8 różnych sposobów, co umożliwia rozwiązanie wspomnianych wyżej zagadnień [6], [7]. Pominięte w opisie dane techniczne (jak wielkość pól informacji, szybkości transmisji między blokami, stosowane technologie i układy itp.) maszyny manchesterskiej można znaleźć w [7], [9], [10]. Ponadto w [10] zaprezentowano interesujące wyniki oceny sprawności maszyny manchesterskiej, porównania szybkości jej działania z innymi systemami, charakterystyki wykonania programów oraz innych parametrów działającego systemu.

Doświadczenia w dziedzinie systemów sterowania przepływowego zgromadzone przez grupę badawczą na Uniwersytecie w Manchester są zachęcające, choć jeszcze wiele problemów z dziedziny oprogramowania i konstrukcji sprzętu wymaga rozwiązania.

LITERATURA

[1] Arvind A., Gostelow K. P.: A Computer Capable of Exchanging Processors for Time. Information Processing, pp. 849-853, North-Holland, Amsterdam, 1977

[2] Catto A. J., Gurd J. R.: Nondeterministic Dataflow Graphs. Information Processing, Proc. 8th IFIP World Computer Congress, pp. 251-256, North Holland, Amsterdam, 1980

[3] Catto A. J., Gurd J.: Resource Managment in Dataflow. Proc. Conf. on Functional Programming Languages and Computer Architecture, Porthsmouth (UK), pp. 77-84, North Holland, Amsterdam, 1981

[4] Comte D., Hifdi N., Syre J. C.: The Data Driven LAU Multiprocessor System Results and Perspectives. Proc. IFIP'80 Congress, pp. 175-180, North Holland, Amsterdam, 1980

[5] Dennis J. B.: Data Flow Supercomputers. Computer, Vol. 13, No. 11, pp. 48-56, 1980

[6] Gurd J., Watson I.: Data Driven System for High Speed Parallel Computing Structuring Software for Parallel Execution. Computer Design, Vol. 9, No. 6, pp. 91-100, 1980

[7] Gurd J., Watson I.: Data Driven System for High Speed Parallel Computing Hardware Design. Computer Design, Vol. 9, No. 7, pp. 97-106, 1980

[8] Gurd J., Glauert J., Kirkham C. C.: Generation of Dataflow Graphical Object Code for the Lapse Programming Language. CONPAR'81, Lecture Notes in Computer Science, Vol. 111, pp. 155-168, Springer-Verlag, Berlin, 1981

[9] Gurd J., Watson I., Glauert J.: A Multilayered Data Flow Computer Architecture. Department of Computer Science, University of Manchester, March 1980

[10] Gurd, J., Kirkham C. C., Watson I.: The Manchester Prototype Dataflow Computer. Communications of the ACM, Vol. 28, No. 1, pp. 34-52, 1985

[11] Marczyński R. W., Milewski J.: Data Driven System Based on a Microprogrammed Processer Module. Proc. 40th Ann. Intern. Symp. on Computer Architecture, SIGARCH Newsletter, Vol. 11, No. 3, pp. 98-106, 1983

[12] Requa J. E., McGraw J. R.: The Piecewise Data Flow Architecture Architectural Concepts. IEEE Trans. on Computers, Vol. C-32, No. 5, pp. 425-438, 1983

[13] Rumbaugh J.: A Data Flow Muliprocessor. IEEE Trans. on Computers, Vol. C-26, No. 2, pp. 138-146, 1977

[14] Sowa M., Murata T.: A Data Flow Computer Architecture with Program and Token Memories. IEEE Trans. on Computers, Vol. C-31, No. 9, pp. 820-824, 1982

[15] Takahashi N., Amamiya M.: A Data Flow Processor Array System-Design and Analysis. Proc. 40th Ann. Intern. Symp. on Computer Architecture, SIGARCH Newsletter, Vol. 11, No. 3, pp. 243-250, 1983

[16] Watson I., Grud J.: A Practical Data Flow Computer. Computer, Vol. 15, No. 2, pp. 51-57, 1982.

Wprowadzenie do grafiki komputerowej I. O. Angella

Grafika komputerowa jest w daszym ciągu, przede wszystkim przedmiotem badań naukowych, a coraz liczniejsze opracowania książkowe są nadal jej opisami od strony prawie wyłącznie teoretycznej – opisami jej matematycznych podstaw. Z tych też powodów są zrozumiałe głównie dla badaczy grafiki i raczej odstraszają niż zachęcają niespecjalistów. Drugie już wydanie monografii I. O. Angella „Wprowadzenie do grafiki komputerowej" (tytuł oryginału „A practical introduction to computer graphics") jest wyjątkiem. Ujmuje bowiem grafikę od strony praktycznej bez zbytniego zagłębiania się w teoretyczną analizę tych zagadnień. Jest próbą zafascynowania czytelnika możliwościami grafiki, programowania efektownych geometrycznych kształtów, wciągnięcia go w technikę programowania (nawet złożonych) rysunków przy użyciu prostych środków. Autor wprowadza bowiem jedynie podstawowe pojęcia z dziedziny grafiki komputerowej. Nie jest to zbyt wyrafinowana grafika,
bez jej opanowania nie można jednak przejść do grafiki zaawansowanej.

Monografia ta, powstała na bazie wykładów dla studentów trzeciego roku kierunków ekonomicznych Uniwersytetu Londyńskiego (jej autor jest wieloletnim wykładowcą grafiki komputerowej w London School of Economics) jest logicznym rozwinięciem podstawowych pojęć grafiki wraz z przykładami ich praktycznego wykorzystania. Dla każdego z podstawowych zagadnień (np. przekształcenia przestrzeni dwuwywmiarowej), w książce znajduje się krótkie wprowadzenie od strony analitycznej (przypomnienie), następnie zaś są podane przykłady programów (ich fragmenty) i generowanych przez nie rysunków. Podane są również wskazania co do rozbudowy (bądź modyfikacji) tych programów w określonym kierunku i efekty graficzne, które ta modyfikacja spowoduje. Tekst książki jest w zasadzie zbiorem opisów i wypisów 72 (krótszych bądź dłuższych) programów grafiki i sugestią ich rozbudowy oraz modyfikacji zgodnie z potrzebami określonego użytkownika. Jest więc zbiorem przykładów, jak można grafikę komputerową wykorzystać w praktyce. Autor sądzi, że po przestudiowaniu jego książki i dogłębnym przeanalizowaniu zamieszczonych w niej przykładów, czytelnik może programować większość zadań grafiki samodzielnie. Takie właśnie podejście do grafiki autor uważa za jedyny sposób na zrozumienie jej zasad. Takie jest również moje zdanie na ten temat.

Książka w żadnym razie nie jest wprowadzeniem do grafiki komputerowej, jak to wynika z tytułu jej polskich wydań, i autorowi (raczej) nie o takie opracowanie chodziło. Wcześniej zostało bowiem wydanych kilka wartościowych monografii na ten temat, np. W. F. Newman, R. F. Sproull, Principles of interactive computer graphics (McGraw-Hill, New York, 1979) czy przetłumaczona ostatnio na język polski książka T. Pavlidisa pt. Grafika i przetwarzanie obrazów. Algorytmy. W tytule polskiego wydania książki I. O. Angella brakuje podkreślenia najistotniejszej cechy książki – brakuje słowa „praktyka" bądź „programowanie", bądź obydwu tych słów równocześnie. Odpowiedniejszym moim zdaniem byłby np. tytuł „Wprowadzenie do programowania grafiki". Książka ta jest bowiem właśnie wprowadzeniem do programowania grafiki (jak to zostało ujęte w tytule oryginału). Jest jedyną monografią z grafiki komputerowej traktującą o podejściu do tych zagadnień od strony użytkowej, bez zbytniego zagłębiania się w analizę teoretyczną. Autor wprowadza jedynie podstawowe pojęcia, które jednak są wystarczające do wykonywania stosunkowo złożonych rysunków. Stawia praktyczne zadania i podaje przykładowe programy ich rozwiązania. Ponadto zachęca czytelnika do eksperymentów na tych programach, by je dogłębnie opanować i tak zmodyfikować, by stanowiły rozwiązania praktycznych zadań czytelników. Sugeruje ponadto, aby na bazie jego książkowych przykładów czytelnik stworzył własną bibliotekę procedur graficznych.

Użyty w książce język opisu zadań programowania grafiki jest (często skrajnie) prosty. Chciałoby się rzec: język nie informatyka a laika w informatyce. Stąd jedyne wymaganie, jakie jest postawione czytelnikowi, to umiejętność programowania w języku Fortran oraz znajomości geometrii analitycznej (w zakresie współrzędnych kartezjańskich) na poziomie ogólnokształcącej szkoły średniej.

dokończenie na s. 30

WOJCIECH MOKRZYCKI
Instytut Podstaw Informatyki
Polska Akademia Nauk
Warszawa

Przegląd technik grafiki komputerowej (2)

W drugiej części artykułu omówiono systemy programowania grafiki komputerowej i zagadnienia normalizacji w systemach graficznych.

SYSTEMY PROGRAMOWANIA GRAFIKI KOMPUTEROWEJ

Nieodzownym warunkiem efektywnego dialogu operatora z komputerem jest odpowiedni system programowania. Z doświadczeń wiadomo, że konwersacyjna grafika komputerowa zdecydowanie komplikuje zagadnienia oprogramowania, gdyż programy grafiki wymagają:
- synchronizacji procesów obliczeniowych oraz częstej obsługi złożonych przerwań;
- rozbudowanych struktur danych graficznych oraz wydajnych procedur manipulowania danymi;
- łatwości i naturalności obsługi oraz dużej wydajności systemu.

Składniki programowe systemu graficznego

Podstawowymi elementami programowymi systemu graficznego są:
- oprogramowanie podstawowe,
- język graficzny,
- zbiory danych graficznych,
- język dyrektyw.

Oprogramowanie podstawowe służy do obsługi transmisji z (lub do) dołączonych do komputera urządzeń graficznych. Oprogramowanie to, dołączone do systemu operacyjnego komputera, jest integralną częścią specjalizowanego (graficznego) systemu operacyjnego.

Języki graficzne służą do tworzenia, wprowadzania i wyprowadzania obrazów (grafika rastrowa) i rysunków (grafika konturowa). Języki te umożliwiają operowanie danymi graficznymi oraz opisywanie wprost obrazów widzialnych przez użytkownika. W porównaniu z innymi językami upraszczają proces przetwarzania danych z atrybutami graficznymi. W projektowaniu języków graficznych istnieją trzy tendencje:
- rozbudowa biblioteki kompilatora istniejącego języka o procedury graficzne,
- rozbudowa istniejącego języka o struktury danych i instrukcje graficzne,
- tworzenie nowych języków graficznych.

Dwie pierwsze tendencje mają tę zaletę, że nie wymagają nauki nowego języka, lecz tylko fragmentów już istniejącego (znanego). Można poza tym wykorzystać dorobek programistyczny wcześniej zgromadzony w bibliotekach procedur użytkowych. Jednak języki te są mało przydatne do oprogramowania systemu o specyficznych właściwościach. Trzecia tendencja wymaga największego nakładu pracy i nauki nowego języka od podstaw. Nie pozwala na łatwe korzystanie z istniejących już podprogramów graficznych, zapewnia jednak narzędzie najefektywniejsze do konkretnego zastosowania.

Zbiory danych graficznych są to zbiory, których struktura zakłada występowanie geometrycznych (takich jak długości, powierzchnie, grubości) i niegeometrycznych (kolory, jaskrawości) oraz innych informacji dotyczących elementów obrazu (np. rodzaj materiału, wytrzymałość). Języki graficzne wraz ze zbiorami danych graficznych umożliwiają budowanie systemów graficznych.

Język dyrektyw jest zbiorem poleceń służących do uaktywniania, za pomocą urządzeń operatorskich, wszelkich programów stosowanych we współdziałaniu z systemem. Polecenie jest wykonywane przez program użytkowy bezpośrednio po odebraniu go od użytkownika i dlatego umożliwia bieżące sterowanie pracą systemu graficznego. Języki dyrektyw należą do specjalnych języków konwersacyjnych i są niezbędnym wyposażeniem konwersacyjnych systemów graficznych.

Struktury zbiorów graficznych

Zasadnicze funkcje systemu graficznego, tj. modelowanie i obrazowanie obiektów fizycznych, analizowanie modeli i wyprowadzanie wyników z tej analizy, tworzenie, modyfikowanie i usuwanie samego modelu – wszystko to wymaga zbiorów danych graficznych, zawierających charakterystyki obiektów, informacje o relacjach zachodzących między tymi obiektami oraz informacje o sposobie i miejscu ich obrazowania. Budowa tych zbiorów jest bardzo ważna, wpływa bowiem w zasadniczy sposób na jakość całego systemu, jego wydajność i wygodę użytkowania. Zbiory danych graficznych mają jedną z najbardziej złożonych organizacji ze wszystkich komputerowych zbiorów danych: nie można bowiem określić z góry ani rozmiaru pamięci zajmowanej przez użytkownika, ani powiązań i relacji ustalonych dopiero w trakcie użytkowania samego systemu. Również parametry fizyczne obiektów (waga, rozmiar, rodzaj materiału itp.) są wprowadzane przez użytkownika zwykle dopiero w trakcie pracy. Tak więc zarówno struktura, jak i zawartość informacyjna zbiorów danych graficznych, mają dla większości zastosowań charakter dynamiczny.

Dane graficzne mają zwykle jedną z czterech następujących struktur: tablicową, wskaźnikową, hierarchiczną lub asocjacyjną.

Struktura tablicowa jest najprostsza; jej podstawowym elementem jest tablica, w której są pamiętane kolejne elementy geometryczne i inne parametry. Stosuje się również struktury wielotablicowe. Struktury tablicowe są proste w użyciu, dzięki czemu proste są także procedury gospodarowania pamięcią. Dlatego szybkość działania procedur pomocniczych i wyszukiwania potrzebnych danych (dostępnych przez indeks tablicy) oraz oszczędność pamięci może być duża. Ograniczeniem jest brak wyróżnienia elementów składowych obiektu – spójne tablice opisują obiekty w całości. Czas usuwania elementów składowych obiektu jest natomiast proporcjonalny do długości tablicy. Jest to typowa **cecha struktury statycznej**, dogodnej w systemach, w których modyfikacje i transformacje modeli i obiektów występują sporadycznie.

Struktury wskaźnikowe są – w odróżnieniu od tablicowych – **strukturami dynamicznymi**, podatnymi na zmiany, ułatwiającymi modelowanie tworzonych dynamicznie obiektów fizycznych. W strukturach tych powiązania między obiektami są specyfikowane za pomocą wskaźników wskazujących kolejne zapisy danych (obiektów). Wskaźniki te umożliwiają powiązanie zapisów danych w różny sposób (np. w ciąg). W zapisie może

wystąpić więcej niż jeden wskaźnik. Każdy z nich wskazuje wówczas zapis należący do innego zbioru lub typu danych. Sekwencyjne powiązania zapisów danych, przy konieczności dotarcia do kolejnego zapisu, wymaga każdorazowo przeglądania ciągu od początku. Czas przeglądania jest więc proporcjonalny do długości ciągu. Z tego powodu stosuje się dwukierunkowy sposób wiązania zapisów wskaźnikami (jeden wskazuje zapis poprzedni, drugi – zapis następny w sekwencji). Powiązanie ostatniego zapisu z pierwszym tworzy **strukturę pierścieniową**. Struktury ze wskaźnikami mają tendencję do zajmowania dużych obszarów pamięci z powodu zwiększającej się liczby wskaźników.

Struktura hierarchiczna, powszechnie stosowana, jest dogodna wówczas, gdy między obiektami zachodzi relacja „wchodzi w skład". Przeglądanie tej struktury może odbywać się w dwóch kierunkach: od góry – szukanie składowych obiektu, od dołu – szukanie obiektu, w skład którego dany obiekt wchodzi. Struktura umożliwia również organizowanie pierścienia zapisów znajdujących się na tym samym poziomie hierarchii.

Struktura asocjacyjna, rzadko obecnie stosowana, umożliwia odwzorowanie powiązań między obiektami o podobnych właściwościach.

NORMALIZACJA W SYSTEMACH GRAFICZNYCH

Dobry system graficzny powinien umożliwiać generowanie odpowiednich środków programistycznych, opartych na pewnej ujednoliconej postaci bazowej języka graficznego. Pozostając systemem uniwersalnym, powinien on mieć także właściwości systemu specjalizowanego, a więc spełniać wymagania wynikające z konkretnych zastosowań. Prowadzone prace nad zbudowaniem uniwersalnego języka graficznego, jako niezbędnego elementu takiego systemu, doprowadziły do kilku ciekawych propozycji w tym zakresie.

W początkowym okresie grafiki komputerowej programowanie odbywało się na poziomie urządzeń. Pierwsze programy graficzne były więc nieprzenośne, co w połączeniu z bardzo dużą czasochłonnością wykonania takiego oprogramowania skutecznie ograniczało powszechne zastosowanie grafiki. Impulsem do zmian było opracowanie taniej technologii produkcji pamięciowych lamp katodowych, zastosowanych do budowy monitorów graficznych. Rozpoczął się drugi okres rozwoju grafiki. Powstały pierwsze propozycje unormowań w dziedzinie oprogramowania graficznego:

- holenderska GPGS,
- angielska GINO,
- grupy IFIP-u SEILLAC,
- grupy GSPC (z ACM SIGGRAPH) CORE.

W propozycji SEILLAC – powstałej na podstawie dwóch poprzednich – zaproponowano rozdzielenie funkcji modelujących systemu (tj. struktury i bazy danych logicznego modelu zadania oraz informacji niezbędnych do komunikacji z systemem graficznym) od funkcji obrazujących. Propozycja CORE była wynikiem osmozy dwóch różnych podejść: dynamicznego i statycznego. W **podejściu dynamicznym** obraz jest przechowywany w szybkiej pamięci i cyklicznie wysyłany z dużą szybkością na ekran monitora konturowego. Taka organizacja systemu jest charakterystyczna dla tzw. **systemów dynamicznych**, w których modyfikacja obrazu dokonuje się w szybkiej pamięci; „reprezentacja obrazu" gra tutaj podwójną rolę w sensie modelu SEILLAC. W **podejściu statycznym**, dostosowanym do innej kategorii sprzętu, obrazy nie są przechowywane w pamięci, ale bezpośrednio po wygenerowaniu wyprowadzane na urządzenie obrazujące (kreślak). Związek między tymi dwoma obrazami stanowi pojęcie „segmentu obrazu", będącego podzbiorem linii i tekstów traktowanym jako niezależna całość. Inny ważny aspekt propozycji CORE to jej szerokie przyjęcie jako normy grafiki konturowej. „Obrazy" składają się z linii niezależnie od tego, czy system jest dynamiczny czy statyczny.

W drugiej połowie lat siedemdziesiątych powstała nowa propozycja: GKS (niemiecka) – *Graphical Kernel System*. Propozycja GKS definiuje – podobnie jak propozycje SEILLAC i CORE na poziomie logicznym – jądro systemu graficznego. W odróżnieniu od propozycji poprzednich jądro to zawiera funkcje m. in. definiowania dwuwymiarowych (**2W**) segmentów i manipulowania nimi oraz funkcje magazynowania, odczytywania i zapisywania informacji graficznej w zbiorach zewnętrznych.

Funkcje systemu GKS są pogrupowane w poziomy zgodne między sobą i mają możliwości rosnące wraz ze wzrostem poziomu. Ponadto GKS może generować obiekty w technice zarówno konturowej, jak i rastrowej.

W końcu lat siedemdziesiątych system CORE zostaje rozszerzony o możliwości operowania obiektami wygenerowanymi w technice rastrowej. Ponadto w ANSI (*American National Standards Institute*) opracowano bardziej ambitną od GKS propozycję hierarchicznego, trójwymiarowego systemu PHIGS (ang. *Programmer's Hierarchical Interactive Graphics System*). Powstała również koncepcja normy dotyczącej poziomu sprzętowego – VDI (ang. *Virtual Device Interface*), określająca zbiór funkcji, które powinny być realizowane w stacji graficznej [2], oraz propozycja VDM (ang. *Virtual Device Metafile*), określająca format zbiorów do przesłań obrazów między różnymi systemami [5].

Prace nad tymi zagadnieniami są prowadzone również w ISO (*International Organization for Standardization*). Na obecnym etapie prac w ISO przyjęto model struktury systemu graficznego, w którym wyróżniono cztery poziomy:
poziom A zawiera bazy danych związane z zastosowaniem; norma sprzęgu dla tego typu danych IGES (ang. *Initial Graphics Exchange Standard*) służy do wymiany danych między różnymi systemami koncepcji;
poziom B zawiera graficzne programy użytkowe; obrazy w tych programach są produkowane z tzw. pierwotników graficznych umieszczonych w biblitece pierwotników graficznych;
poziom C zawiera biblioteki pierwotników graficznych;
poziom D jest znacznie bardziej złożony od poprzednich;
biblioteki pierwotników niezależnych od sprzętu mają podział na część przenośną i nieprzenośną programu; skorzystanie z jakiegoś urządzenia wymaga przejścia przez specjalny sprzęg programowy; projektowany sprzęg zwany VDD (ang. *Virtual Device Driver*) pozwoli uwolnić się od tej warstwy programu.

Do chwili obecnej w ISO zakończono prace i przyjęto jako normę ulepszoną wersję systemu GKS [3] oraz podjęto pracę nad trójwymiarową wersją tego systemu [4]. Ponadto prowadzi się prace nad propozycją PHIGS [1], która najprawdopodobniej zostanie przyjęta jako norma na trójwymiarowy hierarchiczny system graficzny.

Wysuwane są również koncepcje „supersystemu graficznego", który miałby cechy zarówno systemu GKS, jak i PHIGS, tj. precyzyjną definicję (na poziomie logicznym) stacji graficznej i jej sterowania za pomocą funkcji jądra systemu graficznego (jak jest w systemie GKS), ale również zdolność operowania informacją trójwymiarową oraz hierarchizację w segmentacji (jak w systemie PHIGS). System taki powinien mieć i inne cechy, nie występujące w obydwu wymienionych systemach, ale przydatne w praktyce.

W końcu lat siedemdziesiątych również w kraju (w Instytucie Maszyn Matematycznych) podjęto prace nad niezależnym od sprzętu systemem programowania grafiki. Opracowany **system PGS** (Procedury Syntezy Grafiki) został zaprogramowany na komputerze MERA-400, a następnie (jego doskonalsza wersja) również na komputerze SM-4. PGS jest ujednoliconym systemem programowania grafiki komputerowej, którego mechanizmy są niezależne od sprzętu, systemu operacyjego i języków programowania [6]. Występujące z konieczności programy zależne od sprzętu zostały zredukowane do niezbędnego minimum i wydzielone w niezależną warstwę postprocesorów, dopasowujących wewnątrzsystemową strukturę obrazu do sprzętowych funkcji określonego urządzenia graficznego.

Podstawowymi elementami systemu PGS są trzy grupy procedur, tzn. procedury przygotowawcze, generowania obrazu

i postprocesorów oraz zbiór pośredni danych graficznych. Grupa **procedur przygotowawczych** zawiera procedury definiujące oraz procedury transformacyjne. **Procedury definiujące** umożliwiają opisanie pierwotników graficznych na podstawie różnych danych wziętych z programu użytkowego lub wprowadzonych z zewnątrz. **Procedury transformacyjne** umożliwiają wykonywanie podstawowych przekształceń na pierwotnikach graficznych: obrotów, skalowań i przesunięć. Grupa **procedur generowania** obrazu zawiera procedury generujące oraz procedury obsługi zbioru pośredniego danych graficznych. **Procedury generujące** umożliwiają składanie obiektów graficznych z pierwotników graficznych i zapisywanie ich w zbiorze pośrednim danych graficznych. **Procedury obsługi zbioru pośredniego danych graficznych** umożliwiają tworzenie, usuwanie oraz przekształcanie obiektów w nim zapisanych. **Zbiór pośredni danych graficznych** warunkuje niezależność sprzętową systemu PSG. Stanowi wewnątrzsystemowe archiwum obiektów. Jest zbiorem obiektów identyfikowanych przez nazwę, którą nadaje użytkownik, lub przez numer jako kolejny element zbioru. Obiekt w zbiorze pośrednim jest niepodzielnym elementem graficznym, mającym swój układ współrzędnych.

Postprocesory są programami przetwarzającymi obiekty zapisane w zbiorze pośrednim na postać rozkazów i kodów interpretowanych przez poszczególne urządzenia graficzne. W obecnej postaci system PSG jest systemem dwuwymiarowym i nie ma, niestety, możliwości wprowadzania obrazów do komputera. Nie ma również wbudowanego mechanizmu hierarchizacji w segmentacji obrazów.

Przyjęty przez ISO, jako norma, system GKS jest podstawowym systemem graficznym, przewidzianym do większości zastosowań grafiki. Umożliwia on produkowanie, wprowadzanie i wczytywanie do komputera dwuwymiarowych obrazów, przy użyciu urządzeń graficznych obydwu typów: konturowych i rastrowych.

Podstawą koncepcji systemu GKS jest stacja graficzna, złożona z jednego urządzenia graficznego wyjściowego oraz dowolnej liczby graficznych urządzeń wejściowych. Za pomocą jednego programu można sterować wieloma stacjami graficznymi równocześnie. System zawiera funkcje zapisu i odczytu informacji graficznej ze specjalnych zbiorów zewnętrznych. Struktury systemu GKS – całkowicie stablicowane – są generowane zależnie od konfiguracji i przewidywanego zastosowania systemu. W szczególności zawierają pełne dane dotyczące zasobów, konfiguracji, możliwości, aktualnego stanu oraz funkcji systemu realizowanych sprzętowo. Obraz można składać z sześciu typów pierwotników graficznych:

- punktów (znaczników) lub ich ciągów,
- łamanych,
- pól wielokątnych,
- tekstów,
- pól komórkowych,
- uogólnionych pierwotników (o parametrach definiowanych przez użytkownika).

Łamana i pole wielokątne są typowymi pierwotnikami grafiki konturowej, pole komórkowe jest natomiast typowym pierwotnikiem grafiki rastrowej. Pierwotniki mogą być pogrupowane w segmenty, które można magazynować, przesyłać do konkretnej stacji graficznej, poddawać podstawowym transformacjom geometrycznym lub wstawiać w dowolne miejsce programu graficznego. Cechy geometryczne pierwotników (długość, kształt, pole) można zadawać indywidualnie i wówczas dotyczą one wszystkich typów pierwotników i wszystkich stacji. Cechy niegeometryczne (kolor, jaskrawość, tekstura) można zadawać zarówno indywidualnie (każdą cechę), jak i grupowo. W tej ostatniej sytuacji komplet cech odnosi się do pierwotników danego typu i danej stacji.

System zawiera dwa typy transformacji: tzw. transformację normalizacyjną okno-wziernik, przekształcającą współrzędne globalne na współrzędne znormalizowane, oraz tzw. transformację stanowiskową, która przekształca współrzędne znormalizowane na współrzędne konkretnego urządzenia obrazującego.

Będąca w przygotowaniu w ISO trójwymiarowa wersja systemu GKS [4] zawiera również trójwymiarowe wersje sześciu wymienionych typów pierwotników graficznych oraz dodatkowo pierwotniki zbiorów pól wielokątnych dwu- i trójwymiarowe. Jest to obecnie najbardziej dopracowany i najdoskonalszy system programowania grafiki komputerów.

LITERATURA

[1] Computer Graphics – Programmers Hierarchical Interactive Graphics System (PHIGS). Document ISO/TC97/SC 21/N/1405, ISO, Geneva, 17 February 1987

[2] Computer Graphics – Virtual Device Interface. Document ISO/TC97, ISO, Geneva, 23 June 1986

[3] Graphical Kernel System (GKS). Draft International Standard DIS 7942, ISO, Geneva, 23 June 1983

[4] Graphical Kernel System for Three Dimensions (GKS-3D). Functional Description. Document ISO/TC97/SC21/N/853, ISO, Geneva, 14 February 1986

[5] Graphics Virtual Device Matafile. Document ISO/TC97, ISO, Geneva, 14 January 1983

[6] Mokrzycki W.: System programowania grafiki komputerów PSG 2.2. Informatyka, nr 3, 1987.

Uprzejmie informujemy Czytelników, że egzemplarze INFORMATYKI – bieżącą i archiwalne – można kupić nie tylko w kioskach Ruchu, Klubie NOT SIGMY, Zakładzie Kolportażu i Dziale handlowym (szczegóły podano w WARUNKACH PRENUMERATY), ale również

w lokalu naszej redakcji
ul. Mickiewicza 18 m. 17 w Warszawie, tel. 39-14-34
oraz
w specjalistycznej księgarni PP „Domu Książki"
ul. Mokotowska 51/53 w Warszawie, tel. 28-16-14

Zapraszamy wszystkich zainteresowanych.

ATMAN

ZAKŁAD PROJEKTOWO WDROŻENIOWY
Jednostka gospodarki uspołecznionej
04-082 WARSZAWA UL.KRYPSKA 39
TEL. 13-25-61 TLX.812530 agro pl

OFERUJE:
* mikrokomputery zgodne z IBM PC/XT/AT w dowolnych konfiguracjach;
* urządzenia peryferyjne (drukarki, digitizery, plotery, streamery i inne);
* lokalne sieci komputerowe (instalacja u użytkownika);
* materiały eksploatacyjne (dyskietki, kasety do drukarek i streamerów, pisaków do ploterów i inne).

PROJEKTUJE I INSTALUJE:
* systemy informatyczne wspomagania zarządzania i produkcji;
* dołączanie urządzeń nietypowych do IBM PC/XT/AT (urządzeń taśmy papierowej, elektr. maszyn do pisania, nietypowych streamerów, skanerów, przetworników A/C, C/A i innych);
* polskie litery.

ZAPEWNIA:
-bezpłatne doradztwo w zakresie doboru sprzętu i oprogramowania;
-bezpłatne oprogramowanie systemowe i narzędziowe;
-12-miesięczny serwis gwarancyjny;
-odpłatny serwis pogwarancyjny;
-dostawy w terminie 7-14 dni.

TEL.13-25-61 TLX.812530 AGRO PL

EO/869/88

Szanowny Czytelniku!

Aeroklub Warszawski wraz ze spółkami z o.o.
„PC", „PC-ARK" oraz „PROMOTOR" tworzą holding o kierunkach:

elektronika i informatyka oraz działalność turystyczno-sportowa.

O pozycji udziałowców świadczą:

- wielomiliardowe obroty
- ponad 100 osób zatrudnionych w produkcji i usługach związanych z elektroniką
- wiele własnych rozwiązań z zakresu elektroniki, takich jak centrale telefoniczne lub testery urządzeń medycznych
- własne opracowania, takie jak systemy informatyczne dla celów zarządzania, księgowości i nie tylko
- sieć własnych punktów serwisowych sprzętu komputerowego, świadczących usługi gwarancyjne i pogwarancyjne

Jeśli jesteście Państwo zainteresowani współpracą z takim przedsiębiorstwem, jeśli macie ciekawe pomysły w zakresie produkcji lub usług elektronicznych, jeśli interesowałby Was zakup produktów firmy – zapraszamy

WARSZAWA, ul. Irlandzka 9a, ul. Paryska 14
tel.: 17-31-84, 17-41-23, 17-51-10, 17-69-48, teleks: 816962, 817088

WYKAZ PUNKTÓW SERWISOWYCH

1. *Nowy Dwór Maz., Legionowo, Ciechanów, Płock, Bydgoszcz*
   NOWY DWÓR MAZ., ul. Górska 48/6, tel. 75–28–84
   POMIECHÓWEK, ul. Przejazdowa 8, tel. 72-27-00
2. *Konstancin-Jeziorna, Wołomin, Otwock, Grójec, Warka, Warszawa*
   PIASECZNO, Al. Róż 1/20, tel. 56-22-05
3. *Skierniewice, Radomskie, Warszawa*
   STARA IWICZNA, ul. Słoneczna 28, gm. Leszno-wola, tel. 57-02-34
   PIASECZNO, ul. Kusocińskiego 8/73, tel. 33-97-14
4. *Wrocławskie, Wałbrzyskie*
   WROCŁAW, ul. Stalowowolska 5/3, tel. 61-16-00
5. *Leszczyńskie, Kaliskie, Poznańskie, Zielonogórskie, Gorzowskie, Legnickie*
   LESZNO, ul. Janickiego 9, tel. 20-22-44, teleks 045494
6. *Konińskie, Sieradzkie, Poznańskie, Kaliskie*
   KONIN, ul. Okólna 39/4, tel. 246-92
7. *Katowice, Opole, Bielsko-Biała, Częstochowa*
   ZABRZE, ul. Marksa 52/3, tel. 71-17-27
   KNURÓW, ul. M. Konopnickiej 1 B/6, tel. 71-17-27
8. *Jeleniogórskie, Legnickie*
   SZKLARSKA PORĘBA, ul. Broniewskiego 7, tel. 17-28-06
   JELENIA GÓRA, ul. Klonowicza 6/11, tel. 264-24 (kier. 875)
   JELENIA GÓRA, tel. 227-02, 515-09
9. *Łomżyńskie, Ostrołęckie, Białostockie, Suwalskie*
   ŁOMŻA, ul. Spółdzielcza 30/37, tel. 66-67
   ŁOMŻA, ul. B. Prusa 15-63, tel. 686-55
10. *Pilskie, Słupskie, Szczecińskie, Koszalińskie*
    WAŁCZ, Al. Tysiąclecia 12 B/31, tel. 44-81 w. 3775, 40-71
11. *Lubelskie, Radomskie, Bielskopodlaskie, Kieleckie, Tarnobrzeskie, Siedleckie*
    LUBLIN, ul. Nałkowskich 121/20, tel. 46-555
    LUBLIN, ul. Kurantowa 8/30, tel. 71-12-12
12. *Kołobrzeg, Gdańsk, Bydgoszcz, Elbląg, Olsztyn*
    GRUDZIĄDZ, ul. Sobieskiego 52/54, tel. 23-726 (kier. 851)
13. *Toruń, Włocławek*
    GRUDZIĄDZ, ul. Śniadeckich 72/91, tel. 31-131 (kier. 851)
14. *Chełmskie, Zamojskie*
    CHEŁM, ul. Polaniecka 4/54, tel 525-03
15. *Łódzkie, Piotrkowskie, Sieradzkie*
    ŁÓDŹ, tel. 36-19-77 (kier. 042), teleks 885454

EO/962/88

Terminologia systemów operacyjnych (2)

Nawiązując do omówionego w poprzednim numerze terminu polecenie (ang. *command*), warto wspomnieć o pokrewnym zwrocie angielskim *command file*, który oznacza plik tekstowy zawierający zapis ciągu poleceń. Autorzy publikacji nt. systemów operacyjnych używają polskiego odpowiednika – **plik poleceń.** Oczywiście nie jest to błędem, zwłaszcza że wyraz *command* jest w tym zwrocie przetłumaczony poprawnie na **polecenie**. Jednak niektórzy autorzy używają zwrotu – **plik rozkazowy,** który wydaje mi się bardziej odpowiedni jako jednolity termin. Taki wybór nie przeczy bynajmniej zasadzie tłumaczenia terminu *command* na **polecenie.** Ponieważ zwrot *command file* jest samodzielnym terminem, nie obowiązuje nas dosłowne tłumaczenie wyrazów składowych.

Pliki rozkazowe (ang. *command files*) w systemie operacyjnym RSX-11 lub CP/M są oczywiście tym samym, co **pliki wsadowe** (ang. *batch file*) w systemie PC-DOS i tzw. **skrypty** (ang. *scripts*) w systemach Unix i Xenix.

Przy okazji omawiania poleceń nasuwa się również pytanie, jak nazwać po polsku znak lub grupę znaków, a niekiedy nawet cały fragment ekranu, przedstawiany użytkownikowi przez system operacyjny lub inny program jako zgłoszenie do podjęcia współpracy – co określa się po angielsku mianem *prompt*. W starszych systemach operacyjnych, które charakteryzowały się bardzo ograniczonymi możliwościami współpracy z użytkownikiem (sprowadzającymi się do wymiany komunikatów tekstowych), zgłoszenie na ogół załatwiano jednym znakiem, np „⟩", poprzedzając go co najwyżej nazwą programu, jak w systemie RSX-11:

MCR⟩

lub nazwą bieżącego katalogu czy całej ścieżki, jak w systemie DOS:

A:\JANUSZ⟩

Pojedynczy znak „⟩" lub każdy inny występujący w tej roli, jak np. kropka lub gwiazdka, pełni faktycznie funkcję informującą użytkownika (operatora) o gotowości systemu operacyjnego lub innego programu do przyjęcia następnego polecenia, nic więc dziwnego, że określono go mianem znaku gotowości. W praktyce jednak pojedynczy znak występuje niezmiernie rzadko, będąc zazwyczaj ostatnim znakiem wiersza informującego o stanie programu (systemu operacyjnego). Ponieważ całego wiersza nie można nazywać znakiem, w ostatnio wydanych książkach autorzy posługują się terminami **zachęta, zaproszenie** lub **przynaglenie**, uznając, że pojawienie się na ekranie wiersza zakończonego znakiem gotowości stanowi stymulację dla operatora do wydania następnego polecenia. Tak jest istotnie, jednakże w nowszych systemach operacyjnych coraz ważniejszą rolę zaczynają odgrywać elementy informacyjne zawarte w tekście wysyłanym przez program na ekran.

Szczególną cechą nowych systemów operacyjnych jest tzw. **przyjazność** dla użytkownika (ang. *user-friendliness*), polegająca na maksymalnym ułatwieniu użytkowania komputera, co osiąga się przez udostępnienie jak największej ilości ustrukturalizowanej informacji bezpośrednio na ekranie przed wykonaniem polecenia. Oprogramowanie realizujące tę cechę, wraz ze sprzętem ułatwiającym współpracę z komputerem, nazywa się **sprzęgiem użytkowym** (ang. *user interface*). Coraz częściej korzysta się też z pełnej grafiki, szczególnie w tzw. stanowiskach roboczych (ang. *work-stations*), jak Sun, Apollo i in., zapewniając użytkownikom rozbudowany **sprzęg graficzny** (ang. *graphics interface*), z wykorzystaniem myszy, pióra świetlnego, ideogramów, czyli tzw. **ikon** (ang. *icon*) itp. W takim wypadku nie możemy mówić ani o znaku gotowości, ani o wierszu zachęty czy zaproszenia, ponieważ informacja wyświetlona na ekranie i świadcząca o gotowości programu do przyjęcia polecenia, stanowi zazwyczaj pełny wykaz poleceń, czyli tzw. **menu** (ang. *menu*) czynności możliwych do wykonania i zajmuje znaczny

Polskie odpowiedniki angielskich nazw niektórych poleceń

Nazwa angielska	Nazwa polska	Uwagi
abandon	zaniechać	
abort	abortować	Wyraz aborcja istnieje w języku polskim i jest używany w medycynie.
allocate	alokować	Takie tłumaczenie, a nie – przydzielać – jest bardziej precyzyjne.
append	dołączyć	
attach	przyłączyć	
backup	rezerwować	
boot	inicjować wstępnie	
browse	kartkować	
cancel	anulować	
deallocate	dealokować	
delete	usuwać	
dismount	zdemontować	Takie tłumaczenie jest bardziej precyzyjne niż rozmontować.
eliminate	likwidować	
erase	wymazywać	Wyraz ścierać ma zbyt emocjonalne zabarwienie.
free	dealokować	Takie tłumaczenie, a nie – zwalniać – jest bardziej precyzyjne w języku technicznym.
get	pobrać	
halt	zatrzymać	
hello	powitać	
hold	wstrzymać	
include	włączać	
kill	niszczyć	Dosłowny odpowiednik wyrazu *kill* z oczywistych względów nie może być użyty.
login	zarejestrować się	To jest dokładne znaczenie tego pojęcia – rejestrujemy się tak, jak w przychodni.
logout	wyrejestrować się	
mount	zamontować	
put	wydać	Taki wyraz należy przyjąć jako symetryczny do odpowiednika terminu get, ponieważ nie można użyć ani wyrazu wyprowadzać (ang. *output*) ani – zapisywać (ang. *write*)
quit	zaprzestać	
quite	wyciszać	
redirect	przeadresować	
reset	zerować	
restore	odtwarzać	
resume	dokończyć	
save	zachować	
seize	zawłaszczyć	
shutdown	zakończyć sesję	
squeeze	ściskać	
startup	rozpocząć sesję	
stop	stanąć	Nie można użyć wyrazu zatrzymać, bo jest odpowiednikiem terminu *halt*.

fragment ekranu. Najbardziej czytelnym przykładem tego rodzaju dla krajowych użytkowników jest sprzęg użytkowy i tzw. panel podpowiedzi systemu operacyjnego Concurrent DOS (Informatyka, nr 5–6, 1987).

Moim zdaniem, z wymienionych względów, najrozsądniejsze byłoby używanie wyrazu **podpowiedź** jako polskiego odpowiednika angielskiego terminu *prompt*. Stało się to zresztą zauważalne stosunkowo dawno w grafice komputerowej i znalazło wyraz w słowniczku opublikowanym w Informatyce (nr 3, 1985). **Podpowiedź** (ang. *prompt*) jest tam zdefiniowana jako informacja dla użytkownika o czynnościach możliwych do wykonania w następnym kroku: **Podpowiadanie** (ang. *prompting*) jest więc jedną z cech świadczących o tym, że system operacyjny jest przyjazny dla użytkownika (ang. *user-friendly*)

Innym ważnym elementem systemu operacyjnego mającym podstawowe znaczenie dla użytkowników jest zapisany na dysku tekst podręcznika wywoływany na ekran poleceniem noszącym angielską nazwę *help*. Ze względu na znaczenie tego pojęcia i rolę, jaką odgrywa w eksploatacji podręcznik systemu operacyjnego, sądzę, że wyraz *help* można w języku polskim zastąpić tylko wyrazem **samouczek**, ewentualnie zwrotem – **samouczek elektroniczny**. Każde inne tłumaczenie uznałbym za błąd. To prawda, że *help* znaczy po polsku na pomoc, ale tylko wtedy, gdy ktoś potrzebuje pomocy (w języku potocznym). W informatyce sens wykonywania polecenia *help* jest zupełnie inny.

Wywoływanie samouczka jest jednym z wielu poleceń, dla których nie ustalono jeszcze nazw polskich. Ogólna zasada powinna być jednak taka, że nazw poleceń, podobnie jak nazw instrukcji w językach programowania, nie należy tłumaczyć. Trzeba je traktować jako pewien rodzaj nazw własnych, które są z natury nieprzetłumaczalne. Często jednak jest tak, że wyrazy będące nazwami własnymi poleceń występują w tekście informatycznym jako wyrazy pospolite lub terminy – wtedy celowe jest ich przetłumaczenie. W tabeli zgrupowano niektóre zalecane odpowiedniki polskie nazw angielskich dotyczących poleceń, które są trudniej przetłumaczalne bądź w informatyce występują w innym znaczeniu niż znaczenie potoczne (w takich wypadkach nazwie polskiej towarzyszy odpowiednie wyjaśnienie). Bardziej oczywistych i ogólnie przyjętych odpowiedników polskich, jak np. otworzyć – ang. *open*, zamknąć – ang. *close* itp., nie umieszczono w tabeli. Choć nazwy polskie podano w formie bezokolicznika, należy pomiętać, że polecenia powinny być po polsku wydawane w trybie rozkazującym (np. zatrzymaj od zatrzymać, ang. *halt*), a nazwy odpowiednich czynności powinny być rzeczownikami odsłownymi (np. zatrzymanie od zatrzymać).

Janusz ZALEWSKI

Wprowadzenie do grafiki komputerowej I. O. Angella

dokończenie ze s. 24

Polskie tłumaczenie książki ma również inne niedostatki. Niezbyt klarowne i niezgodnie z przyjętą terminologią używa się w książce (bez podania dostatecznie ścisłych definicji) kilku różnych układów współrzędnych (obserwatora, ekranu, aktualnego, absolutnego, zwykłego), które tylko utrudniają zrozumienie omawianych zagadnień. Również terminologia graficzna używana przez tłumacza jest przestarzała, niezgodna z aktualnymi zaleceniami w tym zakresie (propagowanymi np. przez INFORMATYKĘ). Ponadto w tekście książki są i inne (drobne) błędy redakcyjne. Nie umniejszają one jednak pożyteczności tej monografii.

Moim zdaniem książka I. O. Angella z powodzeniem może służyć jako uzupełniający podręcznik akademicki programowania (grafiki w Fortranie) dla studentów początkowych lat wszystkich kierunków studiów oraz dla wszystkich tych, którzy muszą lub mają zamiar wykorzystywać komputer w codziennym życiu. Zwiększa ona zbiór komputerowych narzędzi użytkowych o jeszcze jedno, bardzo pożyteczne i bardzo efektowne – programy grafiki komputerów. Poszerza w ten sposób pole zastosowań komputerów na pewne nowe dziedziny i uatrakcyjnia ich stosowanie w innych.

WOJCIECH MOKRZYCKI

E9/1006/88

ZAKŁAD ELEKTRONIKI KOMPUTEROWEJ
Skr. pocztowa 35, 90-955 Łódź 8, tel. 57-25-83

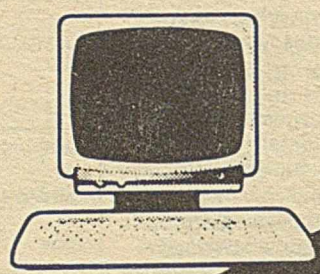
ZEKOM

Oferuje użytkownikom systemów MERA-9150 i REDIFON

TERMINAL EKRANOWY —

MR 1241

będący funkcjonalnym odpowiednikiem monitora MERA-7951.
Terminal MR 1241 przeznaczony jest do wprowadzania danych do systemu minikomputerowego MERA-9150.



JEŚLI INTERESUJE PAŃSTWA:

- *profesjonalny sprzęt o wysokiej jakości, niezawodny w eksploatacji*
- *sprawny serwis*
- *krótkie terminy dostaw lub dostawy natychmiastowe*

TO WYROBY ZEKOMU SĄ DO PAŃSTWA DYSPOZYCJI

dla użytkowników systemów MERA 9150 i REDIFON kompatybilny z MERA 7951

Bardzo przepraszamy Czytelników za wielomiesięczne opóźnienie w wydawaniu INFORMATYKI. Mamy nadzieję, że po zmianie drukarni na Prasowe Zakłady Graficzne RSW „Prasa-Książka-Ruch" w Bydgoszczy uda nam się w ciągu najbliższych miesięcy doprowadzić do regularnego ukazywania się naszego miesięcznika.

Redakcja

Nowe książki

WYDAWNICTWA NAUKOWO-TECHNICZNE

M. Moris Mano „Architektura komputerów". Warszawa 1988, wydanie drugie zmienione, nakład 5000 egz., s. 390, cena 520 zł

W książce podano wiadomości na temat organizacji, projektowania i oprogramowania komputerów. Omówiono jednostki funkcjonalne komputera, organizację centralnego procesora, pamięci oraz urządzeń wejścia-wyjścia. Jest to książka pomocnicza dla studentów wydziału elektroniki, informatyki i automatyki; może być także pomocna inżynierom tych specjalności.

Brian W. Kernighan, Dennis M. Ritchie „Język C". Warszawa 1988, wydanie II, nakład 40 000 egz., s. 256, cena 950 zł

W książce przedstawiono język C, będący nowoczesnym i uniwersalnym narzędziem programowania zrealizowanym na większości mini- i mikrokomputerów. Ze względu na swoją prostotę może być używany do pisania programów numerycznych, przetwarzających teksty lub obsługujących bazy danych. Ważną zaletą języka C jest uniezależnienie go od konkretnego komputera czy systemu operacyjnego. Programy napisane w języku C można więc bez zmian uruchamiać na różnych komputerach z różnymi systemami operacyjnymi.
Książka jest przeznaczona dla programistów, projektantów systemów przetwarzania informacji, pracowników nauki zajmujących się informatyką oraz studentów kierunków informatycznych.

Krzysztof Sacha, Piotr Misiurewicz, Tomasz Kręglewski „Przewodnik po technice mikrokomputerowej". Warszawa 1988, wydanie I, nakład 40 000 egz., s. 128, cena 690 zł

W książce omówiono podstawowe zagadnienia związane z budową, zasadami działania i oprogramowaniem mikrokomputerów: mikroprocesory i ich elementy, pamięci mikrokomputerów, układy sprzęgające, urządzenia zewnętrzne, systemy operacyjne, programy usługowe, wybrane języki programowania, a także zastosowania mikrokomputerów m. in. w systemach czasu rzeczywistego i do sterowania urządzeń.
Książka jest przeznaczona dla szerokiego kręgu czytelników – inżynierów, techników oraz entuzjastów mikrokomputerów.

„Języki mikrokomputerów". Przewodnik dla początkujących pod redakcją Mike'a Ducka; Warszawa 1988, wydanie I, nakład 40 000 egz., s. 176, cena 550 zł

Autorzy tej książki – pedagodzy ze szkół angielskich – wprowadzają młodzież w tajniki programowania mikrokomputerów. Omawiają w przystępny sposób sześć najpopularniejszych języków programowania: Basic, Pascal, Logo, Prolog, Comal i Forth, a także je porównują. Ucząc programowania podają przykłady łatwych i urozmaiconych programów.
Książka jest przeznaczona dla szerokiego kręgu młodych czytelników interesujących się mikrokomputerami.

POLSKIE TOWARZYSTWO INFORMATYCZNE ogłasza drugą edycję OGÓLNOPOLSKIEGO KONKURSU im. JERZEGO TRYBULSKIEGO NA NAJLEPSZE WDROŻONE PRACE Z DZIEDZINY ZASTOSOWAŃ INFORMATYKI W GOSPODARCE NARODOWEJ

Warunki konkursu

1. Prace ogłoszeniowe na konkurs mogą dotyczyć rozwiązań informatycznych wdrożonych w jednostkach gospodarki uspołecznionej i nie uspołecznionej w kraju i za granicą.
2. W konkursie mogą wziąć udział obywatele polscy, którzy są autorami prac wykonanych na terenie kraju.
3. Będą oceniane tylko rozwiązania informatyczne bieżąco eksploatowane przez użytkownika co najmniej jednak rok.
4. Zgłoszone prace będą oceniane według kryterium uzyskanych wymiernych efektów ekonomicznych.
5. Nie będą oceniane wdrożone rozwiązania informatyczne dotyczące: języków programowania, systemów operacyjnych, szeroko rozumianej technologii komputerowego przetwarzania, oprogramowania narzędziowego i pomocniczego.

Wymagania dotyczące zgłoszenia

1. Prawo zgłaszania na konkurs prac spełniających warunki regulaminowe mają dyrektorzy zakładów pracy, kierownicy zespołów autorskich lub eksploatujących oraz indywidualni autorzy rozwiązań informatycznych.
2. Prace konkursowe należy zgłaszać na obowiązującym formularzu.
3. Do karty zgłoszenia należy dołączyć krótką dokumentację rozwiązania oraz opinię bezpośredniego użytkownika zawierającą: czas trwania eksploatacji (w tym datę rozpoczęcia eksploatacji bieżącej), ocenę funkcjonalną, ocenę łatwości eksploatacji (użytkowania), wady i zalety rozwiązania, efekty wymierne i niewymierne, jakie uzyskał użytkownik itp.
4. Przedłożona opinia musi być zatwierdzona przez dyrektora – kierownika jednostki gospodarczej użytkującej system.

Termin i adres nadsyłania zgłoszeń

Dokumentację prac konkursowych, które spełniają wyżej podane warunki i wymagania należy zgłaszać **w terminie do 31 sierpnia 1989 roku pod adresem:**
Polskie Towarzystwo Informatyczne, Oddział Dolnośląski, plac Grunwaldzki 9, p. 15, 50–370 Wrocław, tel. 20-35-80.
Formularze „Karty zgłoszeń" wraz z instrukcją można otrzymać w terenowym Kole PTI lub Oddziale Dolnośląskim. Szczegółowe informacje dotyczące warunków, nagród, organizacji konkursu itp. są podane w regulaminie opublikowanym w numerze 9/85, „Biuletynu PTJ", którymi dysponuje każde terenowe Koło PTI.

Dolnośląski Oddział
Polskiego Towarzystwa Informatycznego
Przewodniczący Komisji Konkursowej
dr Jan Sztajer
(tel. 616061 w. 212, Wrocław)

EO1091/88

Lamport L.: Uproszczone specyfikowanie systemów współbieżnych (1)

INFORMATYKA 1988, nr 10, s. 1

Pierwsza część prezentacji nowej metody ułatwiającej specyfikowanie systemów współbieżnych.

Lamport L.: Simplified concurrent systems specifying (1)

INFORMATYKA 1988, No. 10, p. 1

First part of presentation of a new method which makes specifying concurrent systems easy.

Lamport L.: Vereinfachte Spezifizierung der parallelen Systeme (1)

INFORMATYKA 1988, Nr. 10, S. 1

Erster Teil einer Präsentation von neuer Methode, die Spezifizierung der parallelen Systeme erleichtert.

Sysło M. M.: Maszyny i algorytmy równoległe (1)

INFORMATYKA 1988, nr 10, s. 5

Pierwsza część przeglądu zrealizowanych modeli równoległych systemów komputerowych oraz zaprojektowanych dla nich algorytmów.

Sysło M. M.: Parallel machines and algorithms (1)

INFORMATYKA 1988, No. 10, p. 5

First part of the survey presenting realized models of parallel computer systems and for them designed algorithms.

Sysło M. M.: Parallele Maschinen und Algorithmen (1)

INFORMATYKA 1988, Nr. 10, S. 5

Erster Teil einer Übersicht von realisierten Modellen der parallelen Rechnersysteme und von darüf entworfenen Algorithmen.

Gizdon H., Pawlak A., Wrona W.: VHDL – Ada języków opisu i projektowania sprzętu

INFORMATYKA 1988, nr 10, s. 8

Geneza powstania i charakterystyka języka opisu i projektowania sprzętu komputerowego VHDL, opracowane na zlecenie Departamentu Obrony USA i proponowanego jako standard w tej kategorii języków programowania.

Gizdoń H., Pawlak A., Wrona W.: VHDL – Ada of hardware description and designing languages

INFORMATYKA 1988, No. 10, p. 8

Genesis and characteristics of the VHDL hardware description and designing language, which was elaborated on US Department of Defence order and is expected es standard in this category of programming languages.

Gizdoń H., Pawlak A., Wrona W.: VHDL – eine Ada der Hardwarebeschreibungs- und Projektierungssprachen

INFORMATYKA 1988, Nr. 10, S. 8

Urkunft und Charakteristik der VHDL-Sprache zur Hardwarebeschreibung und – projektierung, die im Auftrage von USA Department of Defence erarbeitet wurde und als Norme für diese Sprachenkategorie vogesehen ist.

Cellary W., Kręglewski J., Maćkowiak R.: System mikrokomputerowy Elwro 800 (1)

INFORMATYKA 1988, nr 10, s. 12

Pierwsza część charakterystyki systemu mikrokomputerowego Elwro 800, zawierająca omówienie rozwiązań architektury, konstrukcji i oprogramowania podstawowego oraz zakresu zastosowań tego systemu.

Cellary W., Kręglewski J., Maćkowiak R.: Elwro 800 microcomputer system (1)

INFORMATYKA 1988, No. 10, p. 12

First part of characteristics of the Elwro 800 microcomputer system, in which architecture, hardware, system software, as well as application range of the system are discussed.

Cellary W., Kręglewski J., Maćkowiak R.: Elwro 800-Mikrorechnersystem (1)

INFORMATYKA 1988, Nr. 10, S. 12

Erster Teil einer Charakteristik von dem Elwro 800--Mikrorechnersystem, der eine Besprechung der Architektur-, Konstruktion- und Systemsoftwarelösungen, sowie des Anwendungsbereiches dieses Systems umfasst.

Kabsch M.: Efektywność systemu dBase III

INFORMATYKA 1988, nr 10, s. 15

Omówienie wyników badań rzeczywistej efektywności mikrokomputerowego systemu zarządzania bazą danych dBase III, umożliwiających oszacowanie efektywności systemu użytkowego we wstępnej fazie projektowania takiego systemu.

Kabsch M.: Effectiveness of the dBase III system

INFORMATYKA 1988, No. 10, p. 15

Discussion of research results of the dBase III system effectiveness, which enables application system effectiveness evaluating in early stage of designing such system.

Kabsch M.: Effektivität des dBase III-Systems

INFORMATYKA 1988, Nr. 10, S. 15

Eine Besprechung von Forschungsergebnissen über Effektivität des dBase III-Datenbankverwaltungssystems, die eine Schätzung von Effektivität eines Anwendungssystems schon in Anfangsstadium seiner Projektierung ermöglichen.

Bielecki J.: Turbo C. Modele pamięci, adresowanie, wskazania adresowe

INFORMATYKA 1988, nr 10, s. 17

Praktyczne wskazówki stosowania niektórych rozwiązań języka programowania Turbo C na mikrokomputerach klasy IBM PC

Bielecki J.: Turbo C. Store models, addressing, address indications

INFORMATYKA 1988, No. 10, p. 17

Practical advices application of the Turbo C language on microcomputers of IBM PC class.

Bielecki J.: Turbo C. Speichermodelle, Adressierung, Adressenanzeigen

INFORMATYKA 1988, Nr. 10, S. 17

Praktische Hinweise der Anwendung von Turbo C--Programmiersprache auf Mikrorechnern der IBM PC--Klasse.

Podraza R.: Sterowanie przepływowe (2)

INFORMATYKA 1988, nr 10, s. 20

Druga część charakterystyki zasad funkcjonowania systemów sterowania przepływowego, zawierająca omówienie implementacji modelu takiego sterowania.

Podraza R.: Dataflow control (2). The model implementation

INFORMATYKA 1988, No. 10, p. 20

Second part of characteristics od data flow control system performing rules, which includes discussion of such control model implementations.

Podraza R.: Durchflusssteuerung (2). Implementierung des Modells

INFORMATYKA 1988, Nr. 10, S. 20

Zweiter Teil einer Charakteristik von Funktionierungsgrundsätzen der Durchflusssteuerungsysteme, der eine Besprechung von Implementierungen solches Steuerungsmedells umfasst.

Mokrzycki W.: Przegląd technik grafiki komputerowej (2)

INFORMATYKA 1988, nr 10, s. 25

Druga część przeglądu technik grafiki komputerowej, zawierająca omówienie systemów programowania tej grafiki oraz zagadnień normalizacji w systemach graficznych.

Mokrzycki W.: Survey of computer graphics technology (2)

INFORMATYKA 1988, No. 10, p. 25

Second part of computer graphics technology survey, which includes discussion of graphics programming systems and standarization problems in graphic systems.

Mokrzycki W.: Eine Übersicht von Techniken der Computergrafik (2)

INFORMATYKA 1988, Nr. 10, S. 25

Zweiter Teil einer Übersicht von Computergrafiktechniken, der eine Besprechung von Gafikprogrammiersystemen und von Normalisierungsproblem in grafischen Systemen umfasst.

Aad van Wijngaarden

Adrian van Wijngaarden, jeden z założycieli IFIP, zmarł 9 lutego 1987 r. w wieku 70 lat.

Wykształcenie inżyniera mechanika zdobył na politechnice w Delft, tam również został uhonorowany tytułem doktora filozofii. Pracę rozpoczął 1 kwietnia 1947 r. w Centrum Matematycznym w Amsterdamie, wkrótce po jego założeniu. Był to początek wspaniałej kariery trwającej do chwili przejścia na emeryturę w 1981 r. Profesor Wijngaarden wyłożył swoją filozofię dotyczącą Centrum Matematycznego w sposób następujący: ktoś musi zajmować się teorią komputerów, musi istnieć osoba pracująca w dziedzinie języków programowania i zawsze ktoś musi zajmować się sprawami sprzętu. Języki naturalne traktował jako dodatkowe wyzwanie.

Pierwsze jego zadanie wykonane dla Centrum Matematycznego polegało na określeniu, na podstawie rozpoznania sytuacji w Stanach Zjednoczonych i Wielkiej Brytanii, przydatności w instytucie matematycznym nowego wynalazku: automatycznych maszyn obliczeniowych. Choć niektórzy wybitni matematycy tamtych lat uważali zainteresowanie komputerami za chwilową fascynację, to Wijngaarden miał wpływ na zmianę ich poglądów i to w sposób zasadniczy. Przekonał zarząd Centrum, że moc obliczeniowa jest zasadniczym czynnikiem postępu w matematyce stosowanej i czynnikiem stymulującym nowo powstającą dziedzinę nauk komputerowych. W rezultacie jego starań powstał komputer o nazwie ARRA (ang. an electric relay machine – elektryczna maszyna przekaźnikowa), przekazany do użytku w 1952 r. przez grupę, na czele której stał van Wijngaarden, a której członkami byli G. A. Blaauw, E. W. Dijkstra, W. L. van der Poel – pionierzy komputeryzacji. Potem zbudowano ARMAC i X1, maszynę w całości tranzystorową.

Pod koniec lat pięćdziesiątych stało się jasne, że konstruowanie komputerów jest zadaniem dla przemysłu. Obowiązującym kanonem pracy dla naukowców stało się opracowywanie nowych koncepcji i idei. Gwałtowny wzrost możliwości komputerów spowodował zwiększone zainteresowanie stworzeniem niezależnego od komputera, uniwersalnego języka algorytmicznego. W tej właśnie dziedzinie zaczął pracować Wijngaarden, po tym kiedy Centrum Matematyczne zrezygnowało z budowania komputerów. Będąc członkiem międzynarodowej grupy, która zaprojektowała Algol 60, Wijngaarden stał się wkrótce głównym architektem Algolu 68. Mimo braku sukcesu handlowego, głębia pojęciowa języka jest zadziwiająca, a wpływy bardzo szerokie – nawet współcześnie. Praca nad Algolem 68 stanowiła duży wkład van Wijngaardena w działalność IFIP-u.

Jako wiceprezydent prekursorskiej Międzynarodowej Konferencji Przetwarzania Informacji w 1959 r., przedstawiciel Holandii do Zgromadzenia Ogólnego (1960–1971), wiceprezydent IFIP (1962–1964), członek zarządu (1967–1970) został wyróżniony w 1974 r. medalem Silver Core. Praca w komitetach technicznych IFIP TC1 (przewodniczący w latach 1967–1974) i TC2 (członek – 1962–1971) prawdopodobnie znaczyła dla niego najwięcej. Odchodząc na emeryturę, jako kierownik Centrum Matematycznego został specjalnie uhonorowany na Międzynarodowym Sympozjum IFIP nt. Języków Algorytmicznych, co było godnym zwieńczeniem kariery.

Spośród wielu zaszczytów, jakich dostąpił prof. Wijngaarden, należy wymienić jego wybór w 1959 r. do Królewskiej Holenderskiej Akademii Nauk, wybór na pierwszego Honorowego Członka Holenderskiego Towarzystwa Komputerowego w 1972 r. i otrzymanie w 1986 r. nagrody IEEE dla pionierów komputeryzacji.

Aad van Wijngaarden zostawił po sobie nie tylko spuściznę naukową. Została po nim pamięć – pamięć o człowieku mądrym i niezwykle sympatycznym.

Oprac. MK
na podst. IFIP Newsletter

Uproszczone specyfikowanie systemów współbieżnych

dokończenie ze s. 4

Formalne znaczenie specyfikacji jest formułą logiki temporalnej o postaci [5]:

$\exists\ f_1, ..., f_n$ takie, że **X**

gdzie **X** jest formułą opisującą dopuszczalne zmiany f_i i g_i. Mówiąc dokładniej, **X** jest formułą, która ogranicza ciąg stanów przyjmowanych przez system, ograniczając wartości funkcji stanu f_i i g_i dla tego ciągu stanów.

Dlaczego kwantyfikator dotyczy tylko wewnętrznych funkcji stanu, a nie funkcji stanu sprzężenia?

Brak kwantyfikatora dla funkcji stanu sprzężenia jest formalnym wyrażeniem faktu, że sprzężenie należy specyfikować na poziomie realizacji. Użycie kwantyfikatora szczegółowego dla wewnętrznych funkcji stanu umożliwia całkowitą dowolność realizacji tych funkcji stanu. Ponieważ funkcje stanu sprzężenia są wolne w specyfikacji (nie skwantyfikowane), te same funkcje stanu muszą występować w realizacji. Wszystko to powinno stać się bardziej zrozumiałe po rozważeniu następnego przykładu, który dotyczy zagadnienia poprawności realizacji.

Opracował
JANUSZ ZALEWSKI

[5] Należy zauważyć, że funkcje stanu sprzężenia g_i są w tej formule zmiennymi wolnymi, których znaczenie omówiono poniżej.

LITERATURA

[1] Alford M. W. et. al. Distributed Systems – Methods and Tools for Specification. Lecture Notes in Computer Science, Vol. 190, Chapter 5, Springer-Verlag, Berlin 1985
[2] Alpern B., Schneider F. B: Defining liveness. Information Processing Letters, Vol. 21, pp. 181-185, October 1985
[3] Guttag J. V., Horning J. J., Wing J. M.: Larch in Five Easy Pieces. Technical Report 5, Digital Equipment Corporation, Systems Research Center, Palo Alfo (CA), July 1985
[4] Hoare C. A. R.: Communicating sequential processes. Comm. of the ACM, Vol. 21, No. 8, pp. 666-677, August 1987
[5] Lam S. S, Shankar A. U.: Protocol verification via projections. IEEE Trans. on Communication, Vol. 10, No. 4, pp. 325-342, July 1984
[6] Lamport L.: „Sometime" is sometimes „not never" – a tutorial on the temporal logic of programs. Proc. Seventh Annual Symposium on Principles of Programming Languages, pp. 174-185, ACM SIGACT-SIGPLAN, January 1980
[7] Lamport L.: Specifying concurrent program modules. ACM Trans. on Programming Languages and Systems, Vol. 5, No. 2, pp. 190-222, April 1983
[8] Lamport L.: What good is temporal logic? R. E. A. Mason (Ed.), Proc. IFIP 9th World Congress, North Holland, Amsterdam, September 1983
[9] Lamport L.: What it means for a concurrent program to satisfy a specification – why no one has specified priority. Proc. Twelfh ACM Symposium on Principles of Programming Languages, pp. 78-83, ACM SIGACT-SIGPLAN, January 1985
[10] Mead C., Conway L.: Introduction to VLSI Systems. Addison-Wesley, Reading (MA), 1980
[11] Wensley J. et al.: SIFT – design and analysis of a faulttolerant computer for aircraft control. Proc. on the IEEE, Vol. 66, No. 10, pp. 1240-1254, October 1987
[12] Wirth N.: Programming in Modula-2. Third edition. Springer-Verlag, Berlin, 1985.

Ceny ogłoszeń

Od 1 lipca 1987 r. obowiązują następujące ceny materiałów reklamowych publikowanych na łamach INFORMATYKI

Ogłoszenia

ogłoszenia czarno-białe, artykuły reklamowe i informacje naukowo-techniczne (biuletyny) zależnie od objętości: cała strona – 50 tys., 3/4 str. – 45 tys., 2/3 str. – 40 tys., 1/2 str. – 35 tys., 1/3 str. – 30 tys., 1/4 str. – 25 tys., 1/8 str. – 20 tys., poniżej 1/8 str. – 200 zł za 1 cm².

ogłoszenia drobne (zależnie od liczby słów) jedno słowo – 50 zł

Dodatki do ceny podstawowej:

za każdy dodatkowy kolor – 30%
za każdy specjalny kolor (nie wynikający z podstawowych kolorów) – 30%
za pełny kolor (grafika wielobarwna, zdjęcia kolorowe) – 120%
za zamieszczenie ogłoszenia na I lub IV stronie okładki – 100%
za zamieszczenie ogłoszenia na II i III stronie okładki – 50%

Zniżki

dotyczą ogłoszeń całkowitych powtórzeń

za ogłoszenia 3–5-krotne – 10%
za ogłoszenia 6–10-krotne – 20%
za ogłoszenia 11-krotne i powyżej – 30%

za artykuły i wkładki reklamowe wykonane przez zleceniodawcę – 40%
za biuletyny i bloki reklamowe – 60%

W uzasadnionych wypadkach stosuje się zniżki specjalne dla ogłoszeń nie będących powtórzeniami – za zgodą Dyrektora – Naczelnego Redaktora Wydawnictwa NOT S
Ponadto Biuro Ogłoszeń świadczy usługi w zakresie wykonywania zdjęć czarno-b
i barwnych oraz nadbitek wkładek reklamowych

Ceny wkładek

wkładka 2 str. o formacie A4
nakład do 500 egz. 20 tys. zł
nakład 500–1000 egz. 35 tys. zł
wkładka 4 str. o formacie A4
nakład 500 egz. 40 tys. zł
nakład 500–1000 egz. 70 tys. zł

Przy wkładkach o nakładzie powyżej 1000 egz. stosuje się wielokrotność powyższyc
a dodatki za kolory oblicza się jak dla ogłoszeń.
Ogłoszenia przyjmowane są przez:
Dział Ogłoszeń i Reklamy WCiKT NOT SIGMA
ul. Świętojerska 5/7, 00-236 Warszawa
adres do korespondencji: skrytka pocztowa 1004, 00-950 Warszawa
telefony: 31-93-65 lub 31-22-21 w. 196 i 291

Warunki prenumeraty na lata 1988–1989

Prenumeratorzy zbiorowi – jednostki gospodarki uspołecznionej, instytucje i organizacje społeczne zamawiają prenumeratę dokonując wpłat wyłącznie na blankiecie „wpłata-zamówienie" (jest to „polecenie przelewu" rozszerzone dla potrzeb Wydawnictwa o część dotyczącą zamówienia).

Blankiety te będą dostarczane dotychczasowym prenumeratom przez Zakład Kolportażu. Nowi prenumeratorzy otrzymają je po zgłoszeniu zapotrzebowania (pisemne lub telefoniczne) w Zakładzie Kolportażu.

Prenumeratorzy indywidualni – osoby fizyczne zamawiają prenumeratę dokonując wpłaty w UPT lub NBP na blankiecie NBP. Na odwrocie wszystkich odcinków blankietu należy wpisać tytuł czasopisma, okres prenumeraty, liczbę zamawianych egzemplarzy oraz wartość wpłaty. Wpłacać należy na konto: NBP III Oddział Warszawa 1036-7490-139-11.

Prenumerata ulgowa – przysługuje wyłącznie osobom fizycznym – członkom SNT, studentom i uczniom szkół zawodowych. Warunkiem prenumeraty ulgowej jest poświadczenie blankietu wpłaty (przed jej dokonaniem) na wszystkich odcinkach pieczęcią Koła SNT, wyższej uczelni lub szkoły. Sposób zamawiania prenumeraty ulgowej jest taki sam jak prenumeraty indywidualnej. W prenumeracie ulgowej można zamówić tylko po jednym egzemplarzu każdego czasopisma.

Uwaga! Miesięcznik „Aura" może być zamawiany w prenumeracie ulgowej również przez uczniów szkół ogólnokształcących.

Prenumerata ze zleceniem wysyłki za granicę – zamawia się tak jak prenumeratę indywidualną. Dodatkowo należy podać na blankiecie wpłaty nazwisko i dokładny adres odbiorcy.

Cena prenumeraty ze zleceniem wysyłki za granicę jest dwukrotnie wyższa.

Wpłaty na prenumeratę przyjmowane są w terminach:

- do 10 listopada na każdy kwartał, I i II półrocze oraz cały rok następny,
- do 28 lutego na II, III i IV kwartał oraz II półrocze,
- do 31 maja na III i IV kwartał oraz II półrocze,
- do 31 sierpnia na IV kwartał.

Zmiany w prenumeracie można zgłaszać pisemnie tylko w wyżej wymienionych terminach.

Informacji o prenumeracie udziela – Zakład Kolportażu Wydawnictwa NOT S
(ul. Bartycka 20, 00–716 Warszawa), skr. poczt. 1004, 00–950 Warszawa, t
-00-21 wew. 248, 249, 293, 297, 299 lub 40-30-86 i 40-35-89.

Egzemplarze archiwalne czasopism – można nabyć za gotówkę w Klubie
Technicznej, Warszawa ul. Mazowiecka 12 (tel. 27-43-65), lub zamówić pis
Zamówienia na egzemplarze archiwalne czasopism przyjmuje: Zakład Kolportażu
Handlowy, 00–950 Warszawa, skr. poczt. 1004 (tel. 40-37-31), na rachunek dla in
lub za zaliczeniem pocztowym dla osób fizycznych.

Cena w 1988 roku

(dotyczy numerów 7–12)

CENA		Prenumerata normalna		Prenumerata ulgow (bez zmiany)	
normalna	ulgowa (bez zmiany)	kwartalna	półroczna	kwartalna	półroc
250 zł	50 zł	750 zł	1500 zł	150 zł	300

Cena w 1989 roku

miesięczna		kwartalna		półroczna		roczna	
normalna	ulgowa	normalna	ulgowa	normalna	ulgowa	normalna	ulg
300 zł	60 zł	900 zł	180 zł	1800 zł	360 zł	3600 zł	72